MASTER THESIS

TITLE: Energy efficiency improvement using presence control through Wi-Fi access management

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Andy Nazario Quispe Cornelio

ADVISOR: Jesus Alcober Segura

DATE: October, 22 2018

Title: Energy efficiency improvement using presence control through Wi-Fi access management

Author: Andy Nazario Quispe Cornelio

Advisor: Jesus Alcober Segura

Date: October 22, 2018

Abstract

Energy efficiency improvement using presence control through Wi-Fi access management

Energy efficiency is a trending topic nowadays, the usage of energy in any kind of environments is vital to allow technology work around the world, Wi-Fi technology is already a deployed solution in most of this environments to allow users connect to the company network or internet in an efficient way.

The energy efficiency on this environments can be achieved through different techniques, on this thesis the proposal to accomplish this task will be by using Wi-Fi technology as a technology to enable presence control mechanism and be able to control different features inside a certain environment mostly inside offices. Information such as energy usage, quantity of users connected, time of connection or disconnection of a given user will be displayed on a graphical interface generated by a management software and it will be implemented inside a virtual machine.

Also this information will the basis to make decisions in order to control efficiently the energy used into the environment in which the access point is operating.

To achieve this on this proposal an API will be developed, this API will extract information from the office access point via SNMP technology using specific OIDs (Object Identifier) from the MIB tree, this information will be analyzed inside the script to make a decision and send a value to the management software. Finally the management software will save this information on a given database and it will display graphical information about the environment on which the monitoring of presence control were performed.

CONTENTS

CHAPTER 1. INTRODUCTION

Recently, the reduction of energy consumption in buildings has increasingly gained interest mainly because of economic advantages and long-term environmental sustainability. The buildings contribute with a usage of energy of 20 – 40 % worldwide [1], in order to reduce the amount of consumption is necessary to apply different mechanisms that allow the efficient usage of energy into building rooms, Wi-Fi technology is a mechanism that is present in most of the building environments, having an internet wireless connection is a must nowadays, since it allows users to work in an efficient way.

On this project we will implement a system based on Wi-Fi technology to control presence, a control management system and an API. We will develop a code in order to show some features of connected users on the management system dashboard, this information will be the basis of the managing control energy efficiency of the proposal.

1.1. Thesis Objectives

The main objectives of this thesis are:

- Detect the presence of people inside building environments through Wii-Fi technology.

- Display information about energy and registered users connected on each environment using a management software.

- Perform accounting regarding the energy and users on each environment.

1.2. Energy Efficiency

Energy efficiency is a trending topic nowadays due to the raise of energy consumption, the development of new technologies such as deployment of general technological infrastructure over the buildings increases its usage, therefore the incremental usage of resources to generate the energy is becoming from years ago a real problem due to heavy environmental impacts and economically impact from the point of view of companies.

An energy consumption study in 2016 indicates that around 75% of Europe buildings are not energy efficient [2], to achieve energy efficiency in new buildings a complex design must be performed using modern construction methods and materials, also technological advances will play an important role on its design. Most of the spaces in buildings such as halls, receptions, conference rooms will require a technical system

to ensure quality and comfort, to achieve this building energy management systems (BEMS) are installed this system will control the environment for users and optimize energy consumption.

To optimize energy consumption sensor devices are deployed along the building in order to detect presence or measure light, in this case will be used a WI-FI hotspot to detect the presence of a user in a certain environment and perform energy management consumption based on the information obtained from the WI-FI hotspot also there will be used a control management system which concentrates all the network infrastructure on one system instead of using a BEMS (Building Energy Management System) as an exclusive control system for energy, this will reduce the deploying of sensors but to increase reliability still can be used some sensors in order to develop a robust system.

1.3. Wireless Technology

Wireless networks allow remote devices to connect without difficulty, independently these devices are a few feet or several kilometers away. And no need to break through walls to pass cables or install connectors. This has made the use of this technology very popular, spreading rapidly. There are many different technologies that differ in the transmission frequency used, speed and range of their transmissions.

Wireless networks can be classified into four specific groups according to the area of application and the signal range [1-3]: Wireless Personal-Area Networks (WPAN), Wireless Local-Area Networks (WLANs), Wireless Metropolitan-Area Networks (WMAN), and Wireless Wide-Area Networks (WWANs). Figure 1.1 illustrates these four categories.

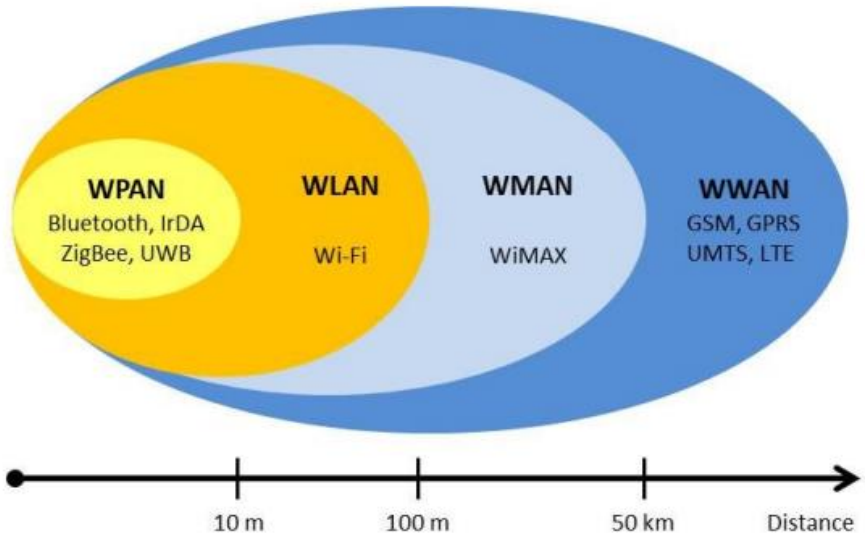


Figure 1.1. Wireless groups by signal range

1.3.1. Wireless Local-Area Network (WLAN)

Wireless Local Area Networks (WLANs) are designed to provide wireless access in areas with a typical range up to 100 meters and, are used mostly in home, school, computer laboratory, or office environments. This gives users the ability to move around within a local coverage area and still be connected to the network. WLANs are based on IEEE 802.11standards, marketed under the Wi-Fi brand name. Due to competition, other standards such as HiperLAN never received much commercial implementation. IEEE 802.11 was simpler to implement and made it faster to the market.

The IEEE 802.11 is a family of different standards for wireless local area networks. The IEEE 802.11b was the first accepted standard, supporting up to 11 Mbps in the 2.4 GHz unlicensed spectrum band. Then, the IEEE 802.11g standard was designed as a higher-bandwidth successor to the IEEE 802.11b. An IEEE 802.11g access point will support 802.11b and 802.11g clients. Similarly, a laptop with an IEEE 802.11g card will be able to access existing 802.11b access points as well as new 802.11g access points. That is because wireless LANs based on 802.11g will use the same 2.4-GHz band that 802.11b uses. The maximum transfer rate for the IEEE 802.11g wireless link is 54 Mbps, but it will automatically back down from 54 Mbps when the radio signal is weak or when interference is detected [3].

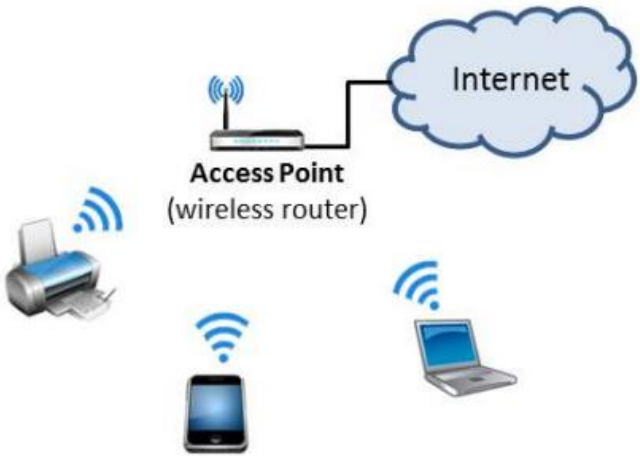


Figure 1.2. Wireless local area network

1.4. Simple Network Management Protocol

The Simple Network Management Protocol (SNMP), issued in 1988, was designed to provide an easily implemented, low-overhead foundation for multivendor network management of routers, servers, workstations, and other network resources. The SNMP specification:

- Defines a protocol for exchanging information between one or more management systems and a number of agents
- Provides a framework for formatting and storing management information
- Defines a number of general-purpose management information variables, or objects

The original version of SNMP (now known as SNMPv1) rapidly became the most widely used vendor-independent network management scheme. However, as the protocol gained widespread use, its deficiencies became apparent. These include a lack of manager-to-manager communication, the inability to do bulk data transfer, and a lack of security. All of these deficiencies were addressed in SNMPv2, issued as a set of proposed Internet standards in 1993.

SNMPv2 has not received the acceptance its designers anticipated. While the functional enhancements have been welcome, developers found the security facility for SNMPv2 too complex. Accordingly, the SNMPv2 working group was reactivated to provide a “tune-up” of the SNMPv2 documents. The result of this effort has been one minor success and one major failure. The minor success is the tune-up of the functional aspects of SNMPv2. The major failure is in the area of security. The working group was unable to resolve the issue, and two competing approaches emerged. With this tune-up, the functional portion of SNMPv2 progressed from proposed to draft Internet standard status as of 1996. Then, in 1997, work began on SNMPv3, which makes additional minor functional changes and incorporates a new security approach [4].

1.4.1. Basic SNMP Concepts

This section examines basic network management concepts that are used as a framework for all three versions of SNMP. We begin with a discussion of the network management architecture, in terms of managed and managing entities, that SNMP is designed to address. Then we look at the protocol architecture used in SNMP. Finally, two important operational concepts, trap-directed polling and proxies, are introduced.

1.4.1.1. Network Management Architecture

The model of network management that is used for SNMP includes the following key elements:

- Management station
- Management agent
- Management information base
- Network management protocol

A management station is typically a standalone device, but may be a capability implemented on a shared system. In either case, the management station serves as the interface for the human network manager into the network management system. The management station will have, at minimum:

- A set of management applications for data analysis, fault recovery, and so on.
- An interface by which the network manager may monitor and control the network. That is, the interface between the user and the network management applications enables the user to request actions (monitoring and control) which are carried out by the management station by communicating with the managed elements of the network.
- A protocol by which the management station and managed entities exchange control and management information.
- A database of information extracted from the management databases of all the managed entities in the network. That is, the management station maintains at least a summary of the management information maintained at each of the managed elements in the network.

Only the last two elements are the subject of SNMP standardization. The other active element in the network management system is the management agent. Key platforms, such as hosts, bridges, routers, and hubs, may be equipped with SNMP agent software so that they may be managed from a management station. The management agent responds to requests for information from a management station, responds to requests for actions from the management station, and may asynchronously provide the management station with important but unsolicited information.

In order to manage the resources in a network, these resources are represented as objects. Each object is, essentially, a data variable that represents one aspect of the managed system. The collection of objects is referred to as a management information base (MIB). The MIB functions as a collection of access points at the agent for the management station; the agent software maintains the MIB. These objects are standardized across systems of a particular class (e.g., bridges all support the same management objects). In addition, proprietary extensions can be made. A management station performs the monitoring function by retrieving the value of MIB objects. A management station can cause an action to take place at an agent or can change the configuration settings of an agent by modifying the value of specific variables.

The management station and agents are linked by a network management protocol, which includes the following key capabilities:

- Get: enables the management station to retrieve the values of objects at the agent.
- Set: enables the management station to set the values of objects at the agent.
- Trap: enables an agent to notify the management station use of objects at the agent objects at the agent of significant events

There are no specific guidelines in the standards as to the number of management stations or the ratio of management stations to agents. In general, it is prudent to have at least two systems capable of performing the management station function, to

provide redundancy in case of failure. The other issue is the practical one of how many agents a single management station can handle. As long as SNMP remains relatively “simple,” that number can be quite high, certainly in the hundreds.

1.5. Network Monitoring Fundamentals

The function of a network monitoring system is to watch the other systems in a computer network for problems. For example, a monitoring system can periodically connect to a web server in a network to ensure that it responds, and if not, send notifications to the network administrators. Although it sounds straightforward, network monitoring systems have grown into complex and sophisticated pieces of software [5].

1.5.1. Concept of Network Monitoring

Network monitoring systems are created to collect data for network management systems. The purpose of network monitoring is the collecting of useful information from various parts of the network so that the network can be managed and controlled using the collected information. Most of the network devices are located in remote locations. These devices do not usually have directly connected terminals so that network management application cannot monitor their statuses easily. Thus, network monitoring techniques are developed to allow network management applications to check the states of their network devices. As more and more network devices are used to build bigger networks, network monitoring techniques are expanded to monitoring networks as a whole.

1.5.2. Functions of Network Monitoring system

The functions of a network monitoring system can be generally classified into three main categories: Performance monitoring, Fault monitoring, and Account monitoring

1.5.2.1. Performance monitoring

Basically, Performance monitoring deals with measuring the performance of the network. There are three important issues in performance monitoring. First, performance monitoring information is usually used to plan future network expansion and locate current network usage problems.

Second, the time frame of performance monitoring must be long enough to establish a network behavior model. Third, choosing what to measure is important. There are too many measureable things in a network. But the list of items to be measured should be meaningful and cost effective. This list of items to be measured is called network indicators because they indicate attributes of the network. Examples of performance metrics to be measured include: network latency, availability, response time, etc.

1.5.2.2. Fault monitoring

Fault monitoring deals with detecting problems in the network. There are two important issues in fault monitoring. First, a network fault can occur at different layers. Thus it is important to know which layers are having problems. Second, fault monitoring requires establishing a normal characteristic of the network in an extended period of time. There are always errors in the network but when there are errors, it does not mean the network is having persistent problems. Some of these errors are expected to occur. For example, noise in a network link can cause transmission errors. The network only has problems, however, when the number of errors increases above its normal behavior. Thus, a record of normal behavior is important.

1.5.2.3. Account monitoring

This function is concerned with observing how users use the network. The network keeps a record of what devices of the network are used by users and how often they are used. This type of information can be used for billing users for network usage, or for non-billing organizations, to predict future network usage.

1.5.3. Data Collection Methods

Typically, a monitoring system will use one of the following methods to collect data about a network element: Agents, Synthetic monitoring, passive monitoring, Logs of activity, Sniffers, and Real user monitoring. In general, for a given type of network element, a particular monitoring technique may be more suitable than the others. A brief description of each technique as well as its advantages and disadvantages follows.

1.5.3.1. The Agent Technique

Most existing network monitoring architectures use this method. A software module called an agent is installed in each monitored device. This software module compiles information about the device in which it resides (for a router, for example, the agent may collect information such as the routing table, the number of error packets received, etc. For a host, the agent may collect information such as CPU utilization, Internet Protocol IP address, etc.). The agent then stores this information in a management database. The software agent is configured to send alerts when it recognizes a problem in the monitored devices. The different agents are configured to send their notifications and alerts to one or more other network devices, known as management stations. An advantage of the agent method is that the management stations are separate from the management agents. This way, a management station will continue to function even if an agent fails, and vice versa. This leads to a generally more robust network monitoring system. Another advantage is that an agent may be programmed to provide virtually every collectable piece of information about the managed device, thereby enabling the monitoring system to monitor almost every aspect of the managed devices and to provide the user with more advanced fault management capabilities.

1.5.3.2. Synthetic monitoring

Sometimes referred to as active monitoring or end to end (E2E) monitoring, the term describes a monitoring mechanism that makes use of a monitored service in the same way a user might. To monitor the status of a service using the E2E approach, a number of behavioral scripts are created, each simulating a single action or navigational path that a typical customer or end-user might take while making use of the service. Those paths are then monitored continuously at specified intervals to measure the availability and response time of the service

1.5.3.3. Passive monitoring

Passive monitoring is a technique used to capture traffic from a network by generating a copy of that traffic, often from a span port or mirror port or via a network tap. Once the data (a stream of frames or packets) has been extracted, it can be used in many ways.

- It can be analyzed in a sniffer.
- It can be examined for flows of traffic, providing information on "top talkers" in a network as well as measures such as TCP round-trip time.
- It can be reassembled according to an application's state machine into end-user activity (for example, into database queries, e-mail messages, and so on.) This kind of technology is common in Real User Monitoring when applied to the http protocol in web applications.
- In some cases, http reassembly is further analyzed for web analytics.

Passive monitoring can be very helpful in troubleshooting performance problems once they have occurred. Passive monitoring differs from synthetic monitoring in that it relies on actual inbound web traffic to take measurements, so problems can only be discovered after they have occurred. While initially viewed as competitive to synthetic monitoring approaches, most networking professionals now recognize that passive and synthetic monitoring are complementary.

1.5.3.4. Real User Monitoring:

Real user monitoring (RUM) is a passive web monitoring technology that records all user interaction with a website. Monitoring actual user interaction with a website is important to website operators to determine if users are being served quickly, error free and if not which part of a process is failing. Software as a Service (SaaS) and Application Service Providers (ASP) use RUM to monitor and manage service quality delivered to their clients. Real user monitoring data is used to determine the actual service-level quality delivered to end-users and to detect errors or slowdowns on web sites. The data may also be used to determine if changes that are promulgated to sites have the desired effect or cause errors.

CHAPTER 2. ANALYSIS AND DESIGN

In the analysis part the components and requirements of the project will be explained and in the part of design we will explain the topology used to develop the solution.

2.1. Analysis

This chapter of the project will determine specifically the software and hardware necessary to develop and the features to deploy a solution that satisfies the objectives of the project.

2.1.1. Requirements

On this part we will explain and analyze each component and specifications of the solution.

2.1.1.1. Access Point with SNMP technology

Since we will work with a software management to control the IT infrastructure it will be necessary to work with access point equipment that can support SNMP technology, this feature will enable the system to gather all the information needed in order to present and control it from the software monitoring system.

In this case we will use a wireless router Linksys WRT54GL with the following specifications:

Linksys WRT54GL		
Network Standards	Ports	Max. Link Rate
IEEE 802.3 IEEE 802.3u IEEE 802.11g IEEE 802.11b	1x 10/100 WAN, 4x 10/100 Switched LAN, 1x Power	54 Mbps

Due to the default Linksys firmware is not enabled to work with SNMP technology we will need to use one firmware which is capable to work with this feature, we have a lot of alternatives. Because of the great community support and the features it enables to manage the network the firmware we are going to use in this case will be DD-WRT which is capable to work with SNMP and is compatible with the specific hardware in our router as explained in the official page https://dd-wrt.com/

The chosen distribution in this case is STD Nokaid Distribution DD-WRT v24-sp2 this information can be find on the official page of DD-WRT project.

2.1.1.2. DD-WRT

DD-WRT is a Linux based alternative Open Source firmware suitable for a great variety of WLAN routers and embedded systems. The main emphasis lies on providing the easiest possible handling while at the same time supporting a great number of functionalities within the framework of the respective hardware platform used [6].

2.1.1.3. Access Control Management Software

There are lots of management software to manage and control IT infrastructure, therefore it is necessary to compare which of them is the best option for us in order to reach and develop the objectives of our project.
It will be analyzed only the main three software that can be used to reach our goals and explain some key features of the chosen software.

2.1.1.3.1. Comparison between monitoring software

Software	API	Support	Graphic Support	Setup
NAGIOS (Free for small environments)	• Enabled by a plugin.	• Large community	• Not easy to implement and customize graphs natively.	• Medium to complex. • Uses third party extensions.
CACTI (GPL license)	• Must work with RRD tool.	• Small to medium size community	• Easy to customize graphs.	• Complex • Uses third party extensions.
ZABBIX (GPL license)	• Enabled natively.	• Medium size community	• Easy to customize graphs natively.	• Medium to complex. • Easy to customize.

From the comparative chart and the features given from each software, the software which offers the best features to implement our solution is Zabbix, since it has a graphical tool that can be customized, API features and also it can run custom monitoring shell scripts, Perl, Python or any other programming language using external checks.

2.1.1.3.2. Zabbix as a monitoring software

Zabbix is software that monitors numerous parameters of a network, health and integrity of servers. Zabbix uses a flexible notification mechanism that allows users to configure e-mail based alerts for virtually any event. This allows a fast reaction to server problems. Zabbix offers excellent reporting and data visualization features based on the stored data. This makes Zabbix ideal for capacity planning.

Zabbix supports both polling and trapping. All Zabbix reports and statistics, as well as configuration parameters, are accessed through a web-based frontend. A web-based frontend ensures that the status of your network and the health of your servers can be assessed from any location. Properly configured, Zabbix can play an important role in monitoring IT infrastructure. This is equally true for small organizations with a few servers and for large companies with a multitude of servers.

Zabbix is free of cost. Zabbix is written and distributed under the GPL General Public License version 2. It means that its source code is freely distributed and available for the general public [7].

2.1.1.3.3. Zabbix features

Zabbix is a highly integrated network monitoring solution, offering a multiplicity of features in a single package [7].

Data gathering	• Availability and performance checks • Support for SNMP (both trapping and polling), IPMI, JMX, VMware monitoring • Custom checks • Gathering desired data at custom intervals • Performed by server/proxy and by agents
Real-time graphing	• Monitored items are immediately graphed using the built-in graphing functionality
Web monitoring capabilities	• Zabbix can follow a path of simulated mouse clicks on a web site and check for functionality and response time
Extensive visualization options	• Ability to create custom graphs that can combine multiple items into a single view
Historical data storage	• Data stored in a database • Configurable history • Built-in housekeeping procedure
Easy configuration	• Add monitored devices as hosts • Hosts are picked up for monitoring, once in the database • Apply templates to monitored devices

Network discovery	• Automatic discovery of network devices • Agent auto registration • Discovery of file systems, network interfaces and SNMP OIDs
Zabbix API	• Zabbix API provides programmable interface to Zabbix for mass manipulations, 3rd party software integration and other purposes.

In this case the distribution we are going to use is zabbix_appliance_3.4.12_x86_64 it can be downloaded from https://www.zabbix.com/download_appliance.

2.1.2. Choosing the Object Identifier (OID)

From here we can begin to work with our solution, we need to use SNMP feature in order to obtain information about the wireless users connected to our network, this will be our starting point to accomplish our goals.
We will work with the Structure of Management Information Version 2 (SMIv2) Management Information Bases (MIBs) installed in the firmware of the access point as part of the SNMP protocol, also it is important to evaluate that the information we are looking for is a unique ID that each device has in order to differentiate the kind of users.

From this evaluation the number or ID we are going to work with is the MAC address of devices since it is a layer 2 global ID for network devices connected to wireless networks. Other way to differentiate users could be using IP addresses but it will be complex and tedious to implement since the IP addresses are assigned using a DHCP server and all IP addresses are shared between users.

From the OID's available on our device regarding the MAC of the users connected we can use the following ones:

2.1.2.1. Private OID

OID (Associated MAC) wlClientMACAddress
'1.3.6.1.4.1.2021.255.3.54.1.3.32.1.4'
{iso(1) identified-organization(3) dod(6) internet(1) private(4)}

This Object (OID) belongs to a "private project" on the MIB tree, so this OID only will work with the DD WRT firmware.
This is a table created from a script and published on this OID, this script is part of the DD WRT firmware and it is already running to set a private OID with the Associated Mac Address clients that are connected in the access point.

The script path is /etc/wl_snmpd.sh and it can be find on the appendix.

This bash script code saves this information into the following OID "1.3.6.1.4.1.2021.255" from console we can obtain some private OID's via SNMP created and updated by this script.

```
appliance@zabbix:/$ snmpwalk -v 2c -c public 192.168.10.1 1.3.6.1.4.1.2021.255
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.1.1 = INTEGER: 1
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.1.2 = INTEGER: 2
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.4.1 = Hex-STRING: 88 C9 D0 F6 BB EC
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.4.2 = Hex-STRING: 54 27 1E B9 61 7D
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.13.1 = INTEGER: -88
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.13.2 = INTEGER: -88
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.26.1 = INTEGER: 58
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.26.2 = INTEGER: 46
```

Figure 2.1. Output of the private OID bulk.

The information we need is placed in the following table '1.3.6.1.4.1.2021.255.3.54.1.3.32.1.4'

```
appliance@zabbix:~$ snmpwalk -v 2c -c public 192.168.10.1 1.3.6.1.4.1.2021.255.3.54.1.3.32.1.4
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.4.1 = Hex-STRING: 54 27 1E B9 61 7D
iso.3.6.1.4.1.2021.255.3.54.1.3.32.1.4.2 = Hex-STRING: 80 01 84 70 58 2E
appliance@zabbix:~$ _
```

Figure 2.2. Wireless users using private OID.

2.1.2.2. Public OID

Another option is to work with the OID of the ARP table where the information placed is related with the Media-dependent `physical' address.

OID (ARP table) ipNetToMediaTable /Public OID
1.3.6.1.2.1.4.22

```
IP-MIB::ipNetToMediaPhysAddress.5014944.192.168.10.105 = STRING: 88:c9:d0:f6:bb:ec
IP-MIB::ipNetToMediaPhysAddress.2147443544.192.168.0.1 = STRING: bc:d1:65:4f:2a:71
IP-MIB::ipNetToMediaPhysAddress.2147443544.192.168.0.25 = STRING: a4:7:b6:f8:be:5
IP-MIB::ipNetToMediaPhysAddress.2147443544.192.168.10.111 = STRING: 54:a0:50:ae:92:d4
IP-MIB::ipNetToMediaPhysAddress.2147443544.192.168.10.140 = STRING: 54:27:1e:b9:61:7d
IP-MIB::ipNetToMediaNetAddress.5016232.192.168.10.105 = IpAddress: 192.168.10.105
IP-MIB::ipNetToMediaNetAddress.2147443544.192.168.0.1 = IpAddress: 192.168.0.1
IP-MIB::ipNetToMediaNetAddress.2147443544.192.168.0.25 = IpAddress: 192.168.0.25
IP-MIB::ipNetToMediaNetAddress.2147443544.192.168.10.111 = IpAddress: 192.168.10.111
IP-MIB::ipNetToMediaNetAddress.2147443544.192.168.10.140 = IpAddress: 192.168.10.140
```

Figure 2.3. Wireless users using public OID bulk.

Now objects are identified by the class OID (1.3.6.1.2.1.4.22) suffixed with the VLAN identifier followed by the IPv4 address, values represent associated MAC addresses in the ARP table, on this table we can see that there are some MACs that don't belong to our network (192.168.10.0/24), this is because on the ARP table all the devices are reflected including other devices connected on the WAN side. In this case the MAC engaged with the IP 192.168.0.1 is our gateway MAC address, in order to filter the

MAC addresses connected on this network will be necessary to develop a custom code to filter this MACs based on the network IP range for each environment.

Also we need to keep in mind that the MACs on this table a user will appear as connected till the ARP timer elapses and mark them as unreachable first and finally erasing them from the table.
We are going to work with both cases, since our goal is to recognize when a user is connected or disconnected in a certain environment and having the possibility to choose from a private or a public OID.

2.1.3. Python and Libraries

We are going to use python 3.5.4 and as an Integrated Development Environment (IDE) we are going to use Spyder 3.2.3.

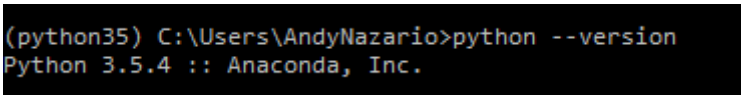


Figure 2.4. Output Python version

The chart shows a description of the main libraries that have been used to develop the Python code.

LIBRARIES	DESCRIPTION
PySNMP	It features fully-functional SNMP engine capable to act in Agent/Manager/Proxy roles [8].
PyZabbix	Is a Python module for working with the Zabbix API, allows you to programmatically retrieve and modify the configuration of Zabbix and provides access to historical data [9].
Zabbix Sender	Zabbix sender is a command line utility that may be used to send performance data to Zabbix server for processing. For sending results directly to Zabbix server or proxy, a trapper item type must be configured [7].

2.2. Design of the Solution

The following graphic is the topology we are going to use for the implementation of the solution.

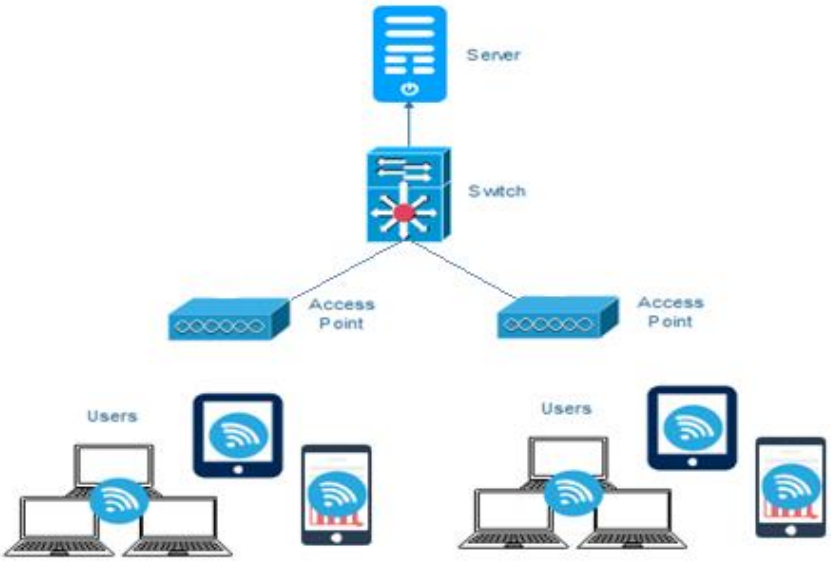
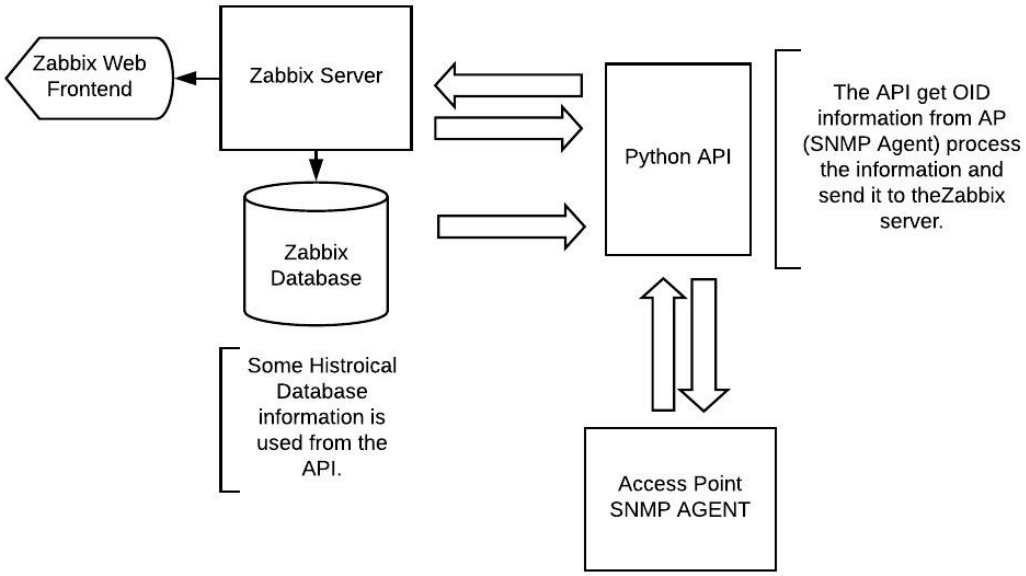


Figure 2.5. Topology of the solution

The server is a virtual machine with a monitoring software in this case Zabbix 3.4.12, the layer 3 switch is used to concentrate all the possible access points connections, route the packets to the internet network and communicate the server with the wireless devices, the users are those devices that will connect to the network to reach internet connection or to access to some resource of the network.



The developed API in Python will be the entity that communicates between the access point getting SNMP information and the Zabbix server sending information to it, in some cases the API will extract historical information from Zabbix server database to analyze it and make decisions. Finally all the information the API sends to the Zabbix server will be displayed as a graphic or chart and it can be monitored from the Zabbix server web frontend via http.

CHAPTER 3. IMPLEMENTATION

On this part will be explained the deployment of each component and also the development of the code in Python for the project.

3.1. Software Installation and Deployment

How we deploy all the software we need to develop our solution will be explained on this part.

3.1.1. Installing DD-WRT on Access Point (Linksys WRT54GL)

To install the new firmware in our device we need to download the firmware in this case we are using dd-wrt.v24_nokaid_generic.bin which allows us to use SNMP technology.

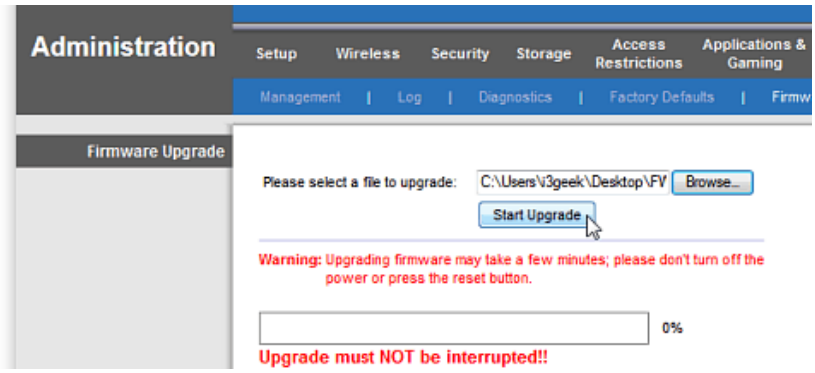


Figure 3.1. DD-WRT upgrade

Once the device has finish the installation this page will be shown, the firmware will ask you for new credentials.

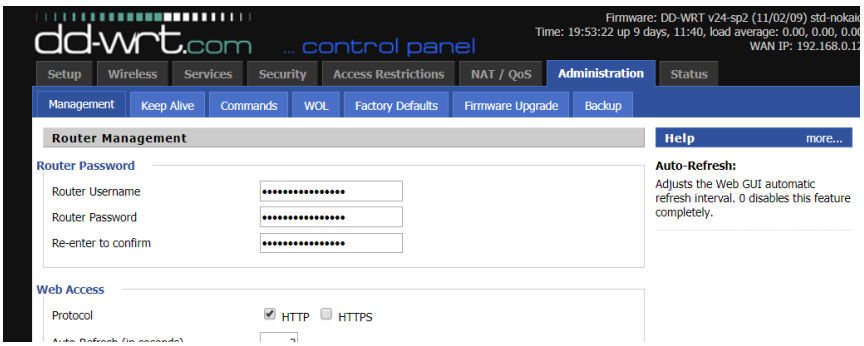


Figure 3.2. Enabling SNMP on DD-WRT

To enable SNMP feature is necessary to go to the tab \services, find SNMP and choose Enable, followed by Apply Settings.

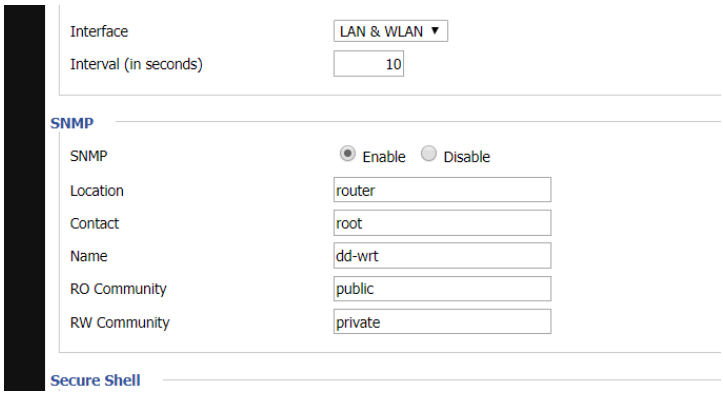


Figure 3.3. Configuring SNMP features

In this case we are going to use the default RO and RW community names.

3.1.2. Installing Zabbix Management System

To install Zabbix we are going to use the appliance zabbix_appliance_3.4.12_x86 version and in order to run the appliance software we are going to use VMware as a platform virtualization for our service. In this case we are using VMware Workstation 14.

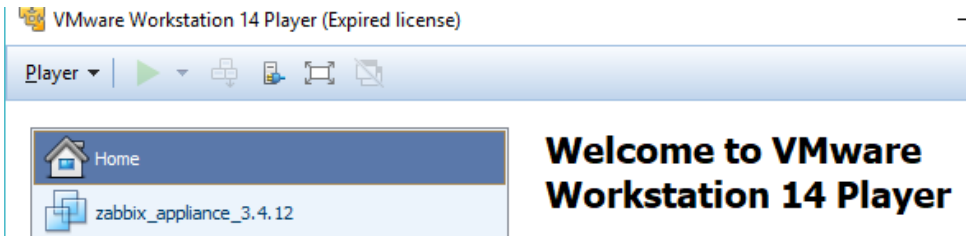


Figure 3.4. Loading appliance with VMware

To run the service we need to find the folder where the .vmdk of Zabbix is saved and run the Virtual Machine.

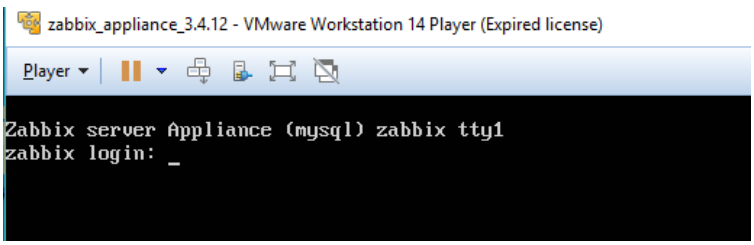


Figure 3.5. Virtual machine running Zabbix appliance

At this point the Virtual Machine is up and working we can enter by CLI or by http management page, to enter by http web management we need to enter the ip address of our VM on the web browser.

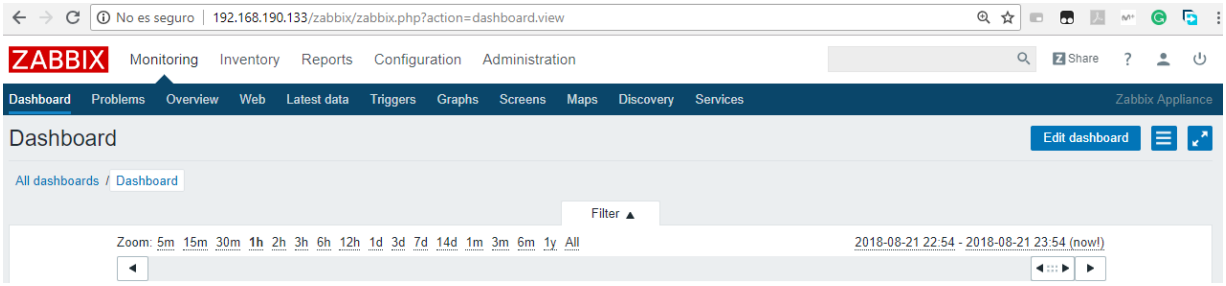


Figure 3.6. Web view of Zabbix appliance.

3.2. Sending Information to Zabbix using API

Our first goal here is to send information to Zabbix related to the energy usage of a given environment in this case an office or classroom, after this information is sent to Zabbix we will need to create a graphic view regarding this information on time.

3.2.1. API Python Code development

3.2.1.1. Energy Usage and Wireless Users Code and deployment

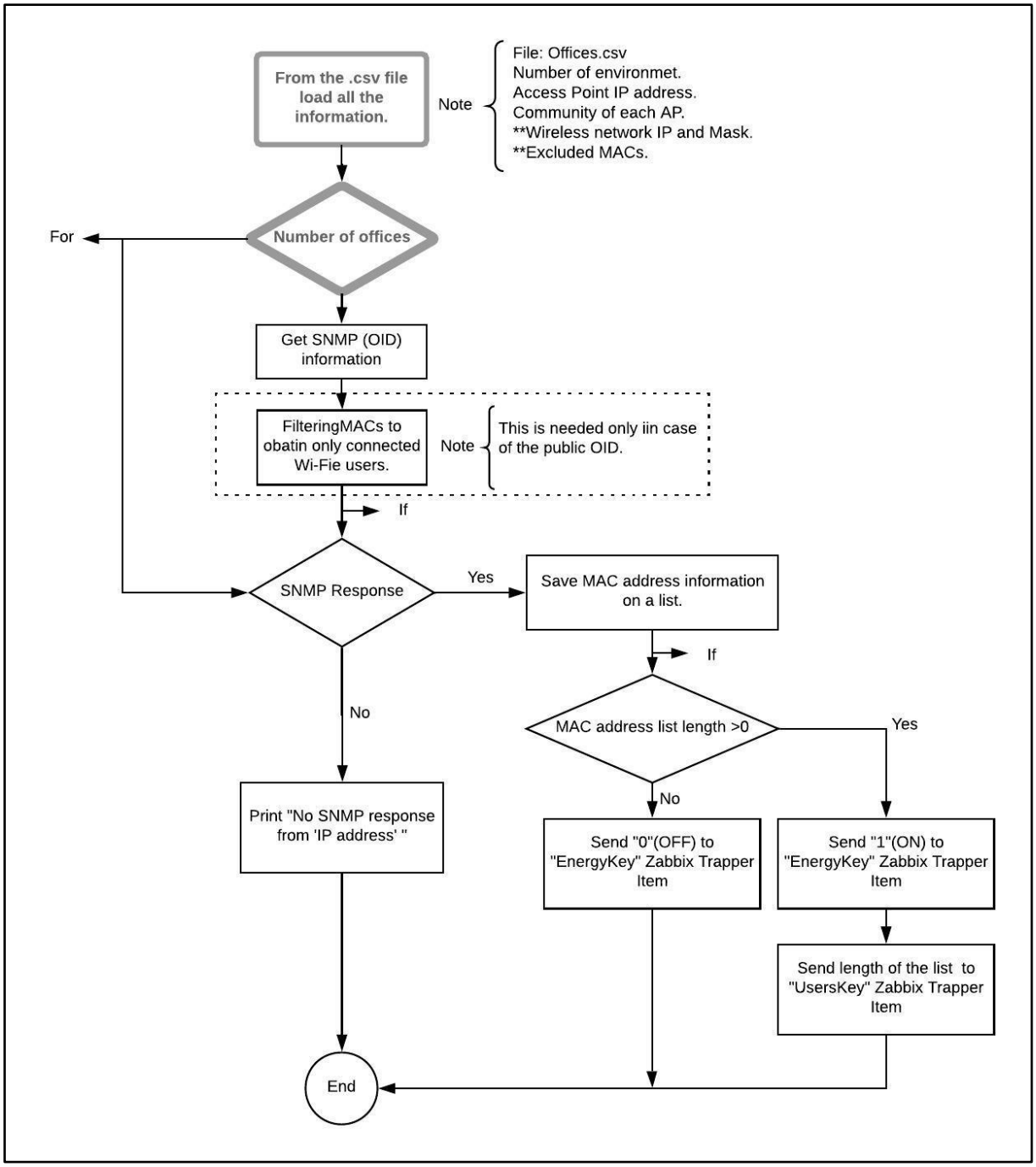
A Python code has been developed in order to send information to Zabbix using Zabbix sender utility. The information is extracted from the access point through SNMP protocol and processed in order to send any wanted information to the Zabbix server using Zabbix sender package and gathered by Zabbix trapper items. Also a CSV file is used to extract information related to the number of offices, IP networks and community of each environment.

	A	B	C	D	E	F
1	Office	IP	community	Excluded MACs	Wireless Clients IP net	Wireless Clients IP netmask
2	1	192.168.10.1	public	0x54a050ae92d4	192.168.10.0	255.255.255.0

Figure 3.7. CSV file used for environments.

Information such network and mask is used to filter the users connected on each environment. Also excluded MACs are used to ignore users that we don't want to detect, on this case a MAC is ignored because it is a wired Ethernet connection, the excluded MACs can be more than one and must be separated by an space between them.

A flow diagram has been created in order to understand the logic of the code:



The code will realize the same for the number of offices written in the CSV file the first functions will obtain the required SNMP OID value based on the IP and the community from the SNMP agent in the access point, this request will return an information.

This information will be processed in order to extract the MACs depending on the type of OID we want to work with, if the OID is private it will only pool the wireless users connected, for the public OID is necessary to use information such as wireless clients net IP and Mask and if needed excluded MAC address. The usage of a private OID will require less compute memory since the OID will return the response directly without

performing too much operations but will only work for devices working with this specific firmware, on the other hand the usage of a public OID will allow us to work with lots of firmware's that can use SNMP feature but it will require much more computing memory and input variables.

Finally the returned value will be the MAC addresses connected.
Basically it will obtain the number of MACs connected to the given network and send a signal (ON "1"or OFF '0') to the corresponding Zabbix trapper item ('EneryKeyOf') created for this matter ,it also will send the length of MACs obtained regarding the number of users connected and send it to another Zabbix trapper item ('UsersKeyOf'). The complete code can be found annexed.

To be able to run our python code inside the Zabbix server is necessary to have python 3.5 installed inside the server.

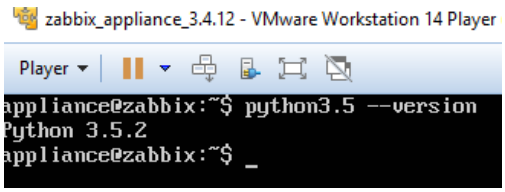


Figure 3.8. Python version inside Zabbix appliance

To run our python code we use External check feature, to achieve this we need to create an 'External Check' item, in this case we have created it inside a 'Global Office' host with the name 'ExternalcheckUsers'.

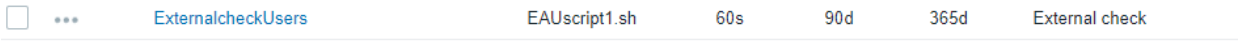


Figure 3.9. "External check" item created.

As well we have created a shell script called EAUscript1.sh to run our python code and placed it inside Zabbix server to be able to run it we need to save it inside the externalscripts folder in this case the path is /usr/lib/zabbix/externalscripts/EAUscript1.sh.

Figure 3.10. Script created to run Python code.

To configure 'ExternalcheckUsers' item we need to write the name of the script saved inside the /externalscripts folder in this case EAUscript1.sh, also we can choose the update interval for this check at any interval.

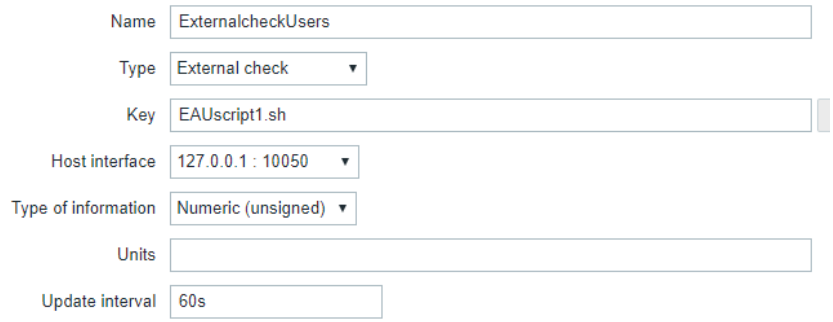


Figure 3.11. “External check” item settings.

Once this is configured Zabbix server will run this script every given seconds in this case we are running only one external check to retrieve information about the energy usage and the number of users connected to the network.

The image is showing python code running every 60 seconds and sending information to the Zabbix trapper items created for this matter.

Office1: EnergyKeyOf1

Timestamp	Value
2018-08-29 18:10:51	1
2018-08-29 18:09:51	1
2018-08-29 18:08:51	1
2018-08-29 18:07:51	1
2018-08-29 18:06:51	1

Office1: UsersKeyOf1

Timestamp	Value
2018-08-29 18:10:51	1
2018-08-29 18:09:51	1
2018-08-29 18:08:51	1
2018-08-29 18:07:51	1
2018-08-29 18:06:51	1

Figure 3.12. Received data on Zabbix trapper items.

At this point we can create a graphical view in order to show the information regarding the energy usage in Zabbix using the “EnegyKeyOF” Zabbix trapper item.

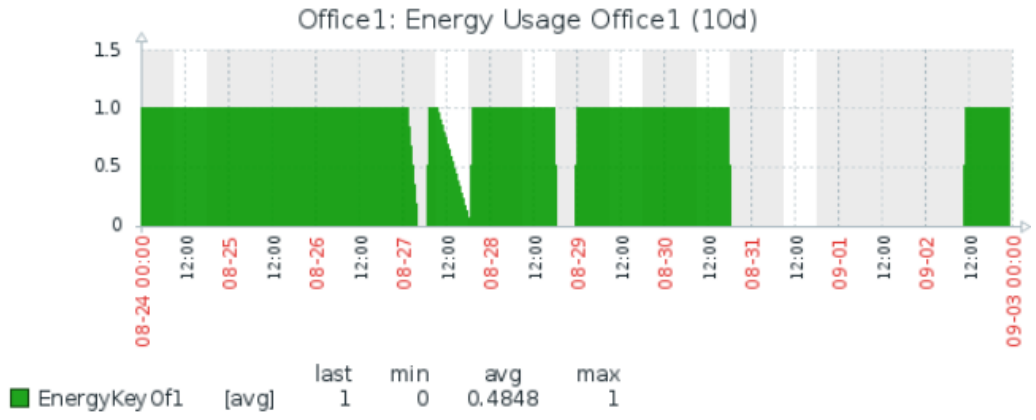


Figure 3.13. Graphic of Energy Usage data

As well we can create a graphical view regarding the number of users using ‘UsersKeyOf’ Zabbix trapper item where all the data is received.

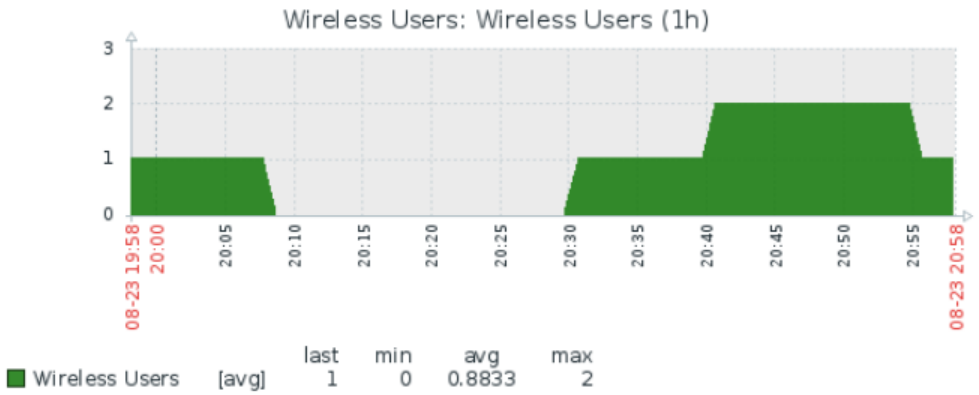


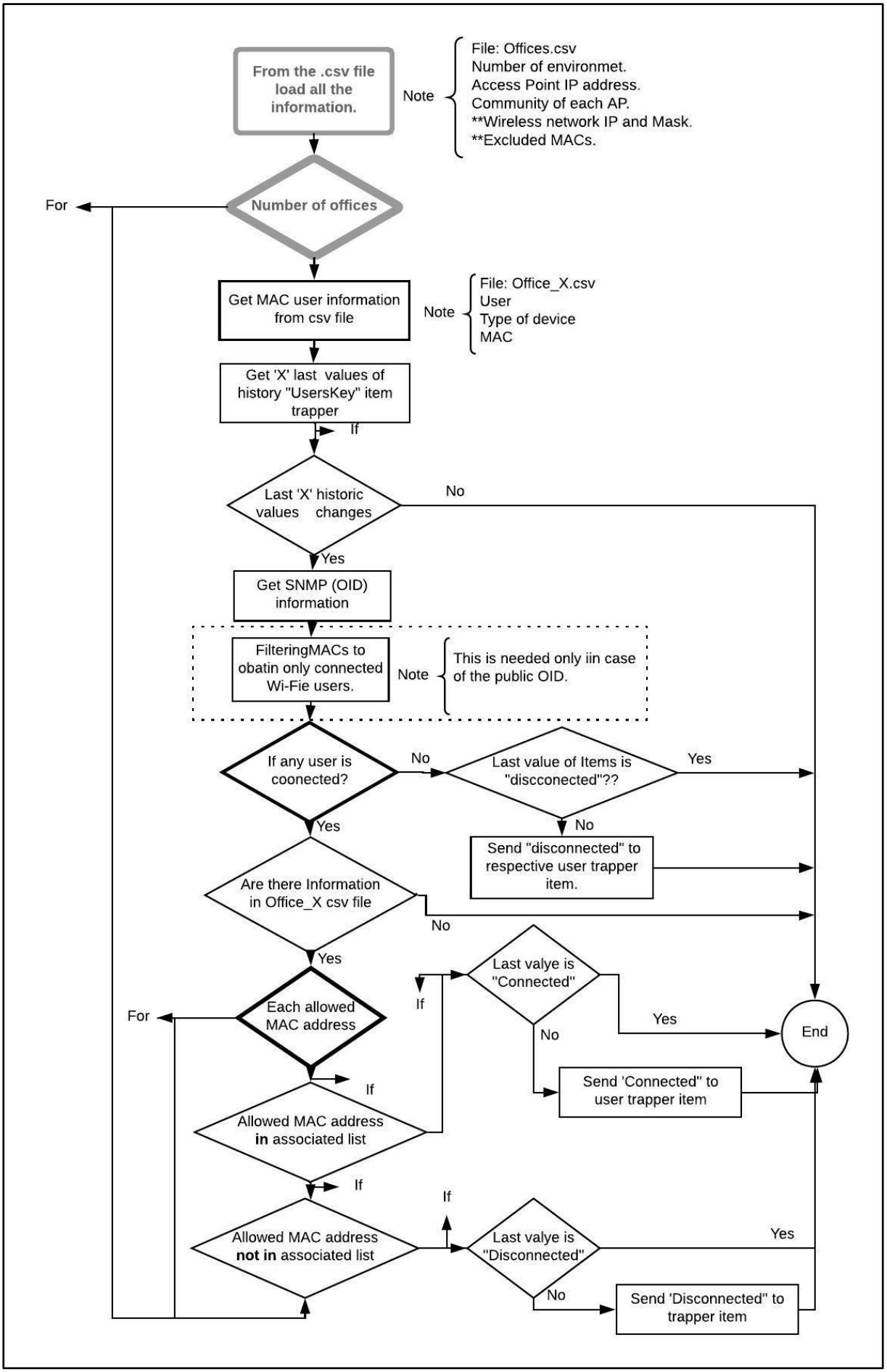
Figure 3.14. Graphic of wireless user’s data.

All this information is stored and its storing can be configured from the web frontend server depending on the application and storing capacity of the server.

3.2.1.2. Allowed Users Access Control Code and Deployment

To achieve the control of the allowed users logged into the wireless network of an environment we have created a Python code, this code will extract information about the users connected to the access point via SNMP protocol. Finally it will compare the MAC of the users connected with a database (CSV file) of allowed users in the network in order to recognize them and enable some features for this “special” users.

A flow diagram has been created in order to understand the logic of the code:


From the .csv file load all the information.
Note
File: Offices.csv
Number of environmet.
Access Point IP address.
Community of each AP.
**Wireless network IP and Mask.
**Excluded MACs.
For
Number of offices
Get MAC user information from csv file
Note
File: Office_X.csv
User
Type of device
MAC
Get 'X' last values of history "UsersKey" item trapper
If
Last 'X' historic values changes
No
Yes
Get SNMP (OID) information
FilteringMACs to obatin only connected Wi-Fie users.
Note
This is needed only iin case of the public OID.
If any user is coonected?
No
Last value of Items is "discconected"??
Yes
No
Yes
Send "disconnected" to respective user trapper item.
Are there Information in Office_X csv file
No
Yes
Last valye is "Connected"
If
For
Each allowed MAC address
Yes
End
No
If
Send 'Connected" to user trapper item
Allowed MAC address in associated list
If
If
Allowed MAC address not in associated list
Last valye is "Disconnected"
Yes
No
Send 'Disconnected" to trapper item

The code will extract information from the Offices.csv file in order to know the information related to each office, once this information is loaded it will analyze historical information about the status of the number of users connected ('UsersKeyOf') to the network if the number of users is the same it will not update the allowed users status inside the office network. Only if the historical information about the connected users change it will update the allowed users information, to achieve this the code will retrieve SNMP information based on the OID we want to work with (private or public), this information will be used to analyze if an allowed users is connected or not and then the code will send an update to the Zabbix trapper item that corresponds to the allowed user, each allowed user will have a Zabbix trapper item linked in order to make easier the control of this users.

For all the Offices we have a CSV file called OfficeX.csv where an index on the column Office will help us to differentiate between offices or environments, this CSV file contains the following information.

Office	USER	POSITION	DEVICE	KEY	MAC
1	Andy Quispe Cornelio	Ingeniero	Laptop	AQC_LAP_OF1	0x54271eb9617d
1	Andy Quispe Cornelio	Ingeniero	Cellphone	AQC_CELL_OF1	0x78c3e91969dc
2	Juanito	Ingeniero	Laptop	JUA_LAP_OF2	

Figure 3.15. CSV file of registered users

To run this code inside Zabbix again we have created an external check item, this item will call a script every given time, and this script must be saved in the externalscripts folder. This script will call our python program in order to run it every certain time.

```
appliance@zabbix:/usr/lib/zabbix/externalscripts$ cat MACallowed.sh
#!/bin/bash
python3 /home/appliance/MACallowed.py
exit
appliance@zabbix:/usr/lib/zabbix/externalscripts$
```

Figure 3.16. Script created to run Python code.

Finally we create the external check item and give it an update interval, this interval will run the script every 180 seconds.

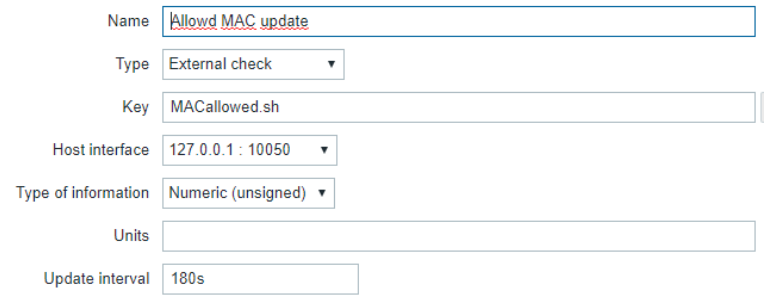


Figure 3.17. “External check” item settings for registered users

The data is stored in the Zabbix trapper item created for each environment.

Office1: USER_AQC_CELL_OF1

Timestamp	Value
2018-10-16 19:54:16	Allowed User Connected Andy Quispe Cornelio with Device Cellphone MAC = 0x78c3e91969dc
2018-10-16 19:51:16	Allowed User Disconnected Andy Quispe Cornelio with Device Cellphone MAC = 0x78c3e91969dc

Figure 3.17. Registered Users on Zabbix Dashboard

CHAPTER 4. TEST

On this part will be explained some problems that arose while testing the system and also a tool used for aggregating, analyzing, visualizing the data generated on the system.

4.1. Problems

Problems are presented along with a way to solve in order to increase the reliability of the system.

4.1.1. Update OID due to a bug on private OID

One of the problems we have found when using the private OID for DD-WRT is that the associated MAC information when no one is connected to the network is not updated, the last user connected always appears as an associated MAC to the Access Points even if no user is connected. We realized that this feature needed to be updated in order to give the correct information, this is a bug that can be resolved using different techniques.

One of the techniques to solve this is using Journaled Flash File System (JFFS), which is used to save user programs and data inside the Access Point flash memory, with this feature we can create an easy script to update snmpd.conf process inside the access point firmware, But in our case the memory available to create a JFFS partition were not possible because all the memory were already used.

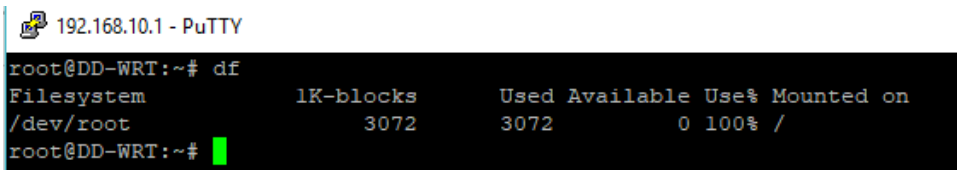


Figure 4.1. Memory available on DD-WRT firmware.

Another option we could use was to work with the rc_startup variable and snmpd_conf, which are directly saved inside the flash memory of the access point. Creating a script which also creates another script and run it from the snmpd_conf variable. This procedure is well explained on thesis research called “EXTENSIÓN DE LAS CAPACIDADES DE MONITORIZACIÓN DE JFFNMS EN ENTORNOS WLAN” [10].

Finally the option we have implemented on this work is using the cron feature which allows us to run tasks or commands in a given time or period, there is a cron_jobs variable inside the flash memory we can work with in order to update the SNMP information inside the access point.
For our solution we need to run the following command ‘kill all –HUP snmpd’ to update SNMP information. To achieve this we can use CLI or HTTP management interface.

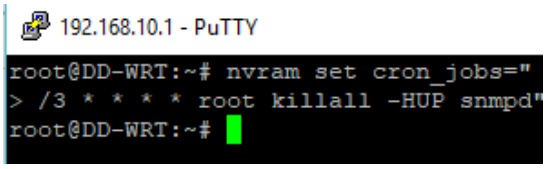


Figure 4.2. NVRAM set Cron command

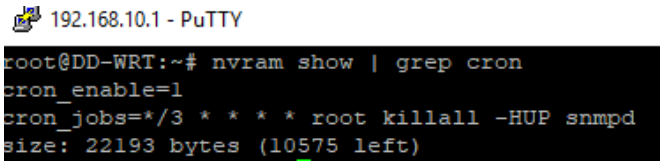


Figure 4.3. NVRAM Cron command

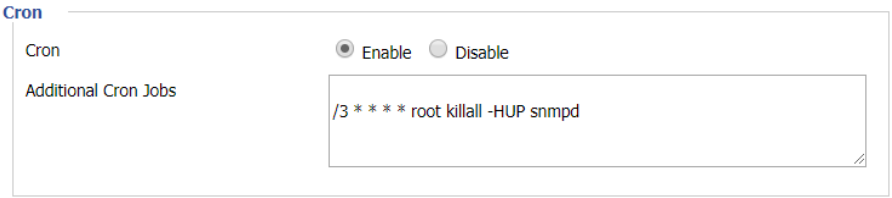


Figure 4.4. NVRAM Cron command via http

Using the command "/3 * * * * root killall -HUP snmpd" we are updating snmpd daemon every 3 minutes to update SNMP information and avoid false positive information.

4.1.2. Presence detection on mobiles

Another problem we have found during this project is that users who use smartphones and tablets tend to appear disconnected until the user stops using the internet connection for a given time and the device changes to suspended mode, this is a feature that mobiles, tablets and some laptops has. They put in a sleep mode its wireless interface in order to save energy this is called PSM (Power Saving Mode). This is a drawback to take into account when a presence detection wants to be achieved, to solve this some additional configuration should be performed, for laptops running windows some power saving mode settings can be changed:

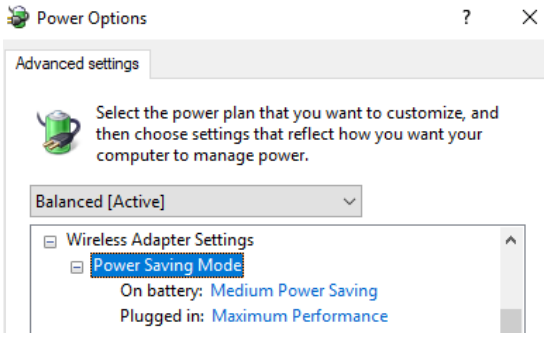


Figure 4.5. Wireless settings on Windows[18]

Power Saving Mode	Description
Maximum Performance	Achieve maximum wireless performance and no power savings.
Low Power Saving	Achieve minimum power savings.
Medium Power Saving	Balance between performance and power savings based on network traffic.
Maximum Power Saving	Achieve maximum power savings.

For cellphone and tables running Android:

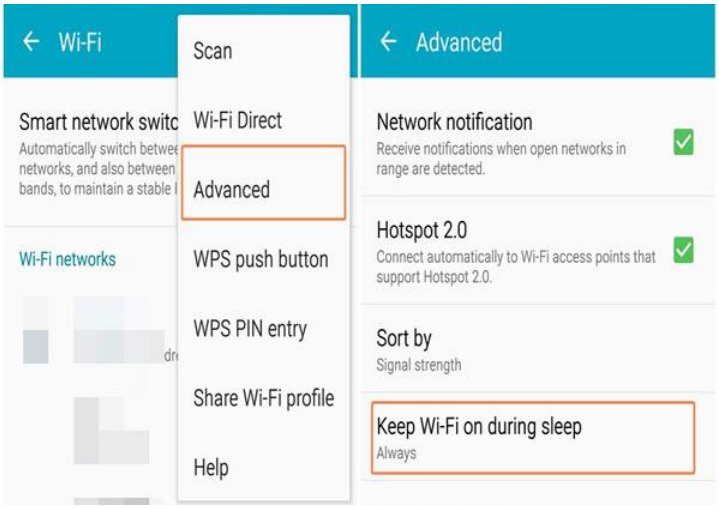


Figure 4.6. Wireless settings on Android [18]

Also disabling the power saving mode feature will keep the WI-FI connection.

4.1.3. Scalability

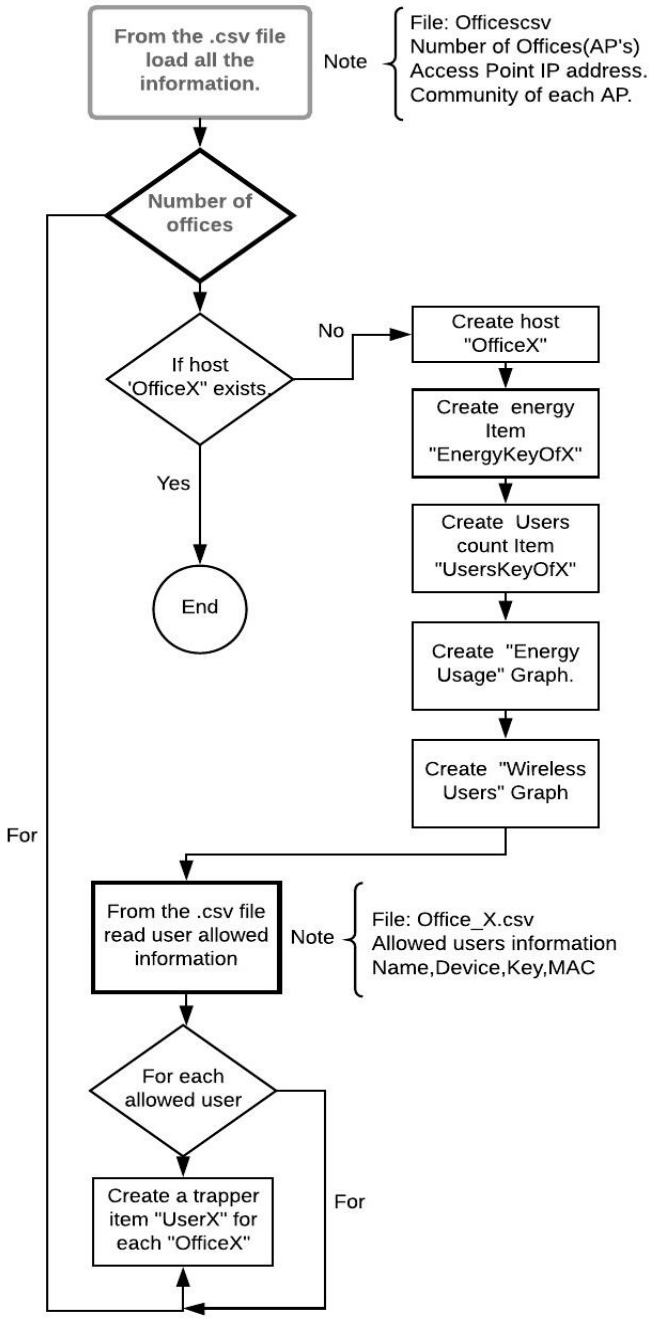
In terms of scalability for this project we have developed a python code in order to create and manage access points, therefore CSV files has been created. The file we have created is Offices.csv on this file we have the ID of the Access Point, ip address and community.

	A	B	C
1	Office	IP	community
2	1	192.168.10.1	public
3	2	192.168.20.1	public
4	3	192.168.30.1	public
5	4	192.168.40.1	public

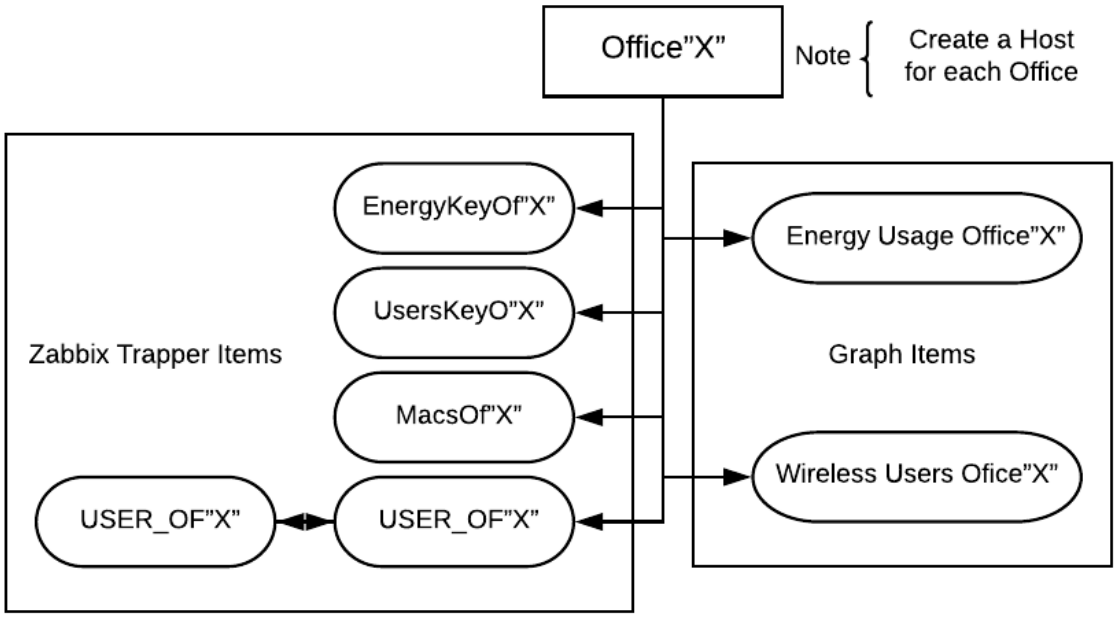
Figure 4.6. CSV file with Offices information.

The python code we have developed will read this file and extract SNMP information based on the OID, ip address and the community, then the information we need to be send to Zabbix will be send to the item trapper of each office based on its ID(Office).

To be able to manage all the hosts and item creation inside Zabbix we have develop a code that will read the Offices.csv file in order to create them:



Based on the information written on the CSV file the code will create:



X is the number of office written in the CSV file.

On Zabbix we can see the creation of the HOSTS, in this case 4 offices from the CSV file.

Office1	Applications 2	Items 14	Triggers 5	Graphs 2	Discovery	Web	192.168.10.1: 10050
Office2	Applications	Items 3	Triggers	Graphs 2	Discovery	Web	127.0.0.1: 10050
Office3	Applications	Items 3	Triggers	Graphs 2	Discovery	Web	127.0.0.1: 10050
Office4	Applications	Items 3	Triggers	Graphs 2	Discovery	Web	127.0.0.1: 10050

Figure 4.7. Creation of Offices from Python code using the API.

Creation of the items (Zabbix trapper):

Wizard	Name ▲	Triggers	Key	Interval	History	Trends	Type
•••	AllowedUsersOf2		AllowedUsersOf2		90d		Zabbix trapper
•••	EnergyKeyOf2		EnergyKeyOf2		90d	365d	Zabbix trapper
•••	UsersKeyOf2		UsersKeyOf2		90d	365d	Zabbix trapper

Figure 4.8. Creation of Zabbix trapper items

Creation of the graphs:

Name ▾	Width	Height
Wireless Users Ofice2	900	300
Energy Usage Office2	900	300

Figure 4.9. Creation of graph items

The automatic creation of this hosts, items and graphs makes easier the scalability of the network, if we have a reasonable quantity of environments to manage this python script will save a lot of time in terms of deployment.

4.2. Managing data tool

Data generated is stored inside Zabbix server, on this case each Zabbix trapper item created stores the data sent by the Python script through the API. The amount of data increases directly proportional with time anjd depending on the interval configured to run the script. The data generated can be large and Zabbix is not prepared to transform this data in order to give it value and obtain relevant information from it.

To solve this problem a tool called Power BI has been used along with the API, this tool is a suite of business analytics tools to analyze data. Since Power BI supports python code to extract information it has been possible to extract data from Zabbix using the API with a Python code.

This software can be downloaded on the official Microsoft page: https://powerbi.microsoft.com/

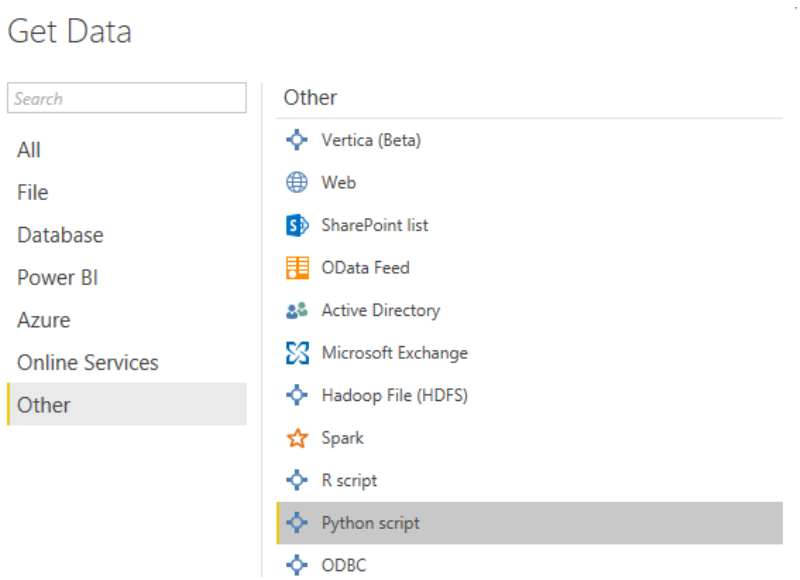


Figure 4.10. Power BI python script support.

The Python script used to extract the data from Zabbix can found in the appendix part of this document.

Since we have created items for each environment it is easy to differentiate the retrieved data, the data can be retrieved by host or by item. Using the API script we have been able to retrieve data through a query regarding energy and users as showed in the following screenshot.

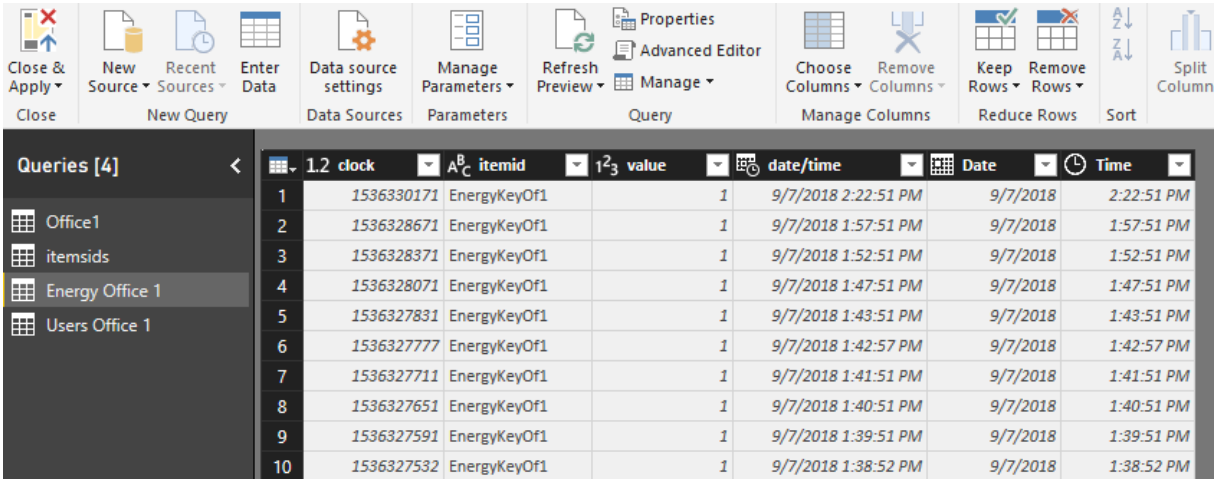


Figure 4.11. Retrieving energy data from Zabbix

The data is retrieved with a clock column on which timestamp is represented, using power BI has been easy to deal with this since it has a tool to convert this data into date/time values as it can be seen in the previous image.

As an example of data analysis we have created a graph with data generated along three days using the system, using this tool we have created a graph that shows the usage of energy along this three days inside an environment (Office).

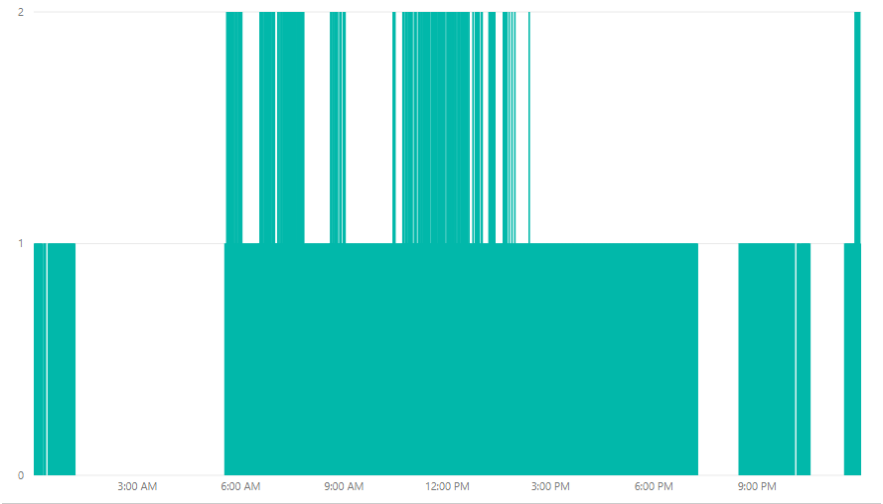


Figure 4.12. Energy usage along three days.

It can be seem that the usage of energy along those three days is grater from 6:00 AM to 3:00 PM and there is no usage at all of energy on this environment from 1:00 AM to 6:00 AM.

Another graph we have created on this tool is regarding the users on this case along these three days, we have the possibility to create a graph that will show us the maximum quantity of users along these days.

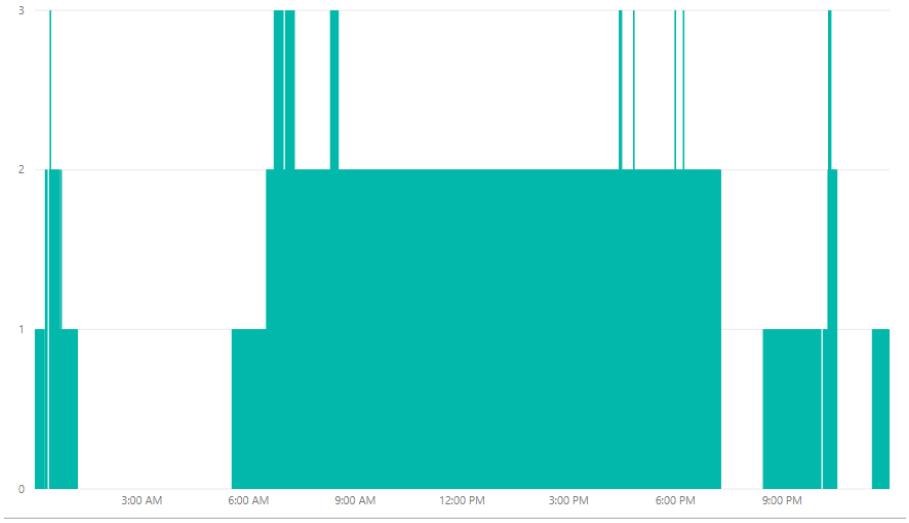


Figure 4.13. Maximum users along three days.

The possibility to analyze data and create graphics using this tool is huge, the data can be analyzed using python and R and also the graphs can be created using this kind of programming code.

CONCLUSIONS

- One of the main goals of this work was to detect people using WIFI technology, this has been the starting point of this work. The detection of people depends exclusively on the wireless connection of user devices, some features like energy saving mode of these devices can obstruct its detection as explained on the test part of the work. This can be overcome by performing some additional configurations improving this way the detection of through this devices. It has to be taken into account that the implementation of features like energy saving by these companies on these devices have been made in order to save energy thus the increase of battery life.

- The second goal of the work was to display information regarding the energy and users on each environment, these has been accomplished by using Zabbix as a management software and an API along with a Python code. A custom python code has been developed in order to accomplish this and the generated data is stored on Zabbix server and showed on the dashboard that can be created for each environment.

- Stored data regarding energy and users can be retrieved by the API, in this case to achieve, transform and present data regarding the accounting has been used a Power BI software that allows retrieve data using python, this software is powerful and can manage thousands of data in order to obtain relevant information from the generated data as presented in the test part of this project.

- It has been probe the concept of using WIFI technology along with SNMP to improve energy efficiency on environments, data generated and analyzed properly can be used to make decisions and improve dramatically the efficiency on given environments.

- Future works can integrate other variables such as power transmission or a piece of hardware to increase the reliability of the system. From the data generated on this system also it can be improved the way the data is analyzed by connecting directly the SQL server from Zabbix with the Power BI software and creating custom scripts to present relevant information and improve the energy efficiency of a given environment.

ACRONYMS

API	Application Programming Interface
ARP	Address Resolution Protocol
ASP	Application Service Providers
BEMS	Building Energy Management System
CLI	Command Line Interface
CSV	Comma Separated Values
E2E	End to end
GPL	General Public License
JFFS	Journaled Flash File System
LAN	Local Area Network
MAC	Medium Access Control
MIB	Management Information Base
OID	Object Identifier
RUM	Real User Monitoring
RRD	Round Robin Database
SaaS	Software as a Service
SMIv2	Management Information Version 2
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
WAN	Wide Area Network
WLANs	Wireless Local-Area Networks

REFERENCES

[1] M. Aksoezen, M. Daniel, U. Hassler, N. Kohler, “Building age as an indicator for energy consumption”, [Online], Available: https://www-sciencedirect-com. [Accessed May 2018]

[2] A. Rudzki, Z. Paciorkiewicz, T. Augustyniak, “Energy Consumption in Office Buildings”, [Online], Available: https://www.skanska.pl [Accessed May 2018]

[3] J. Salazar, “Wireless networks”, [Online], Available: https://upcommons.upc.edu [Accessed May 2018]

[4] W. Stallings. “SNMP and SNMPv2: The Infrastructure for Network Management”, [Online], Available: http://www.hit.bme.hu [Accessed May 2018]

[5] A. Bashir, “University of Khartoum Network Monitoring and Management System”, [Online], Available: http://khartoumspace.uofk.edu [Accessed May 2018]

[6] “DD-WRT”, [Online], Available: https://dd-wrt.com [Accessed May 2018]

[7] “Zabbix Documentation 4.0”, [Online], Available: https://www.zabbix.com [Accessed May 2018]

[8] “PySNMP”, [Online], Available: https://github.com [Accessed June 2018]

[9] “PyZabbix”, [Online], Available: https://github.com [Accessed June 2018]

[10] L. Luceño Forné, Universitat Oberta de Catalunya, “Extensión de las Capacidades de Monitorización de JFFNMS en Entornos wlan”, [Online], Available: http://openaccess.uoc.edu/ [Accessed July 2018]

[11] Microsoft, [Online], Available: https://support.microsoft.com [Accessed August 2018]

APPENDIX

Title: Energy efficiency improvement using presence control through Wi-Fi access management

Author: Andy Nazario Quispe Cornelio

Advisor: Jesus Alcober Segura

Date: October 22, 2018

Python Codes

1. DD-WRT SNMP private OID script

```
#!/bin/sh

place=".1.3.6.1.4.1.2021.255"

refresh() {

  id=1
  lastid=0
  noise_reference=$(wl noise | cut -d" " -f3)

  for mac in $(wl assoclist | cut -d" " -f2)
  do
    if test $lastid -eq 0
    then
      getnext_1361412021255="$place.3.54.1.3.32.1.1.1"
      getnext_136141202125535413321 1="$place.3.54.1.3.32.1.1.1"
      getnext_136141202125535413321 4="$place.3.54.1.3.32.1.4.1"
      getnext_136141202125535413321 13="$place.3.54.1.3.32.1.13.1"
      getnext_136141202125535413321 26="$place.3.54.1.3.32.1.26.1"
    else
      eval getnext_136141202125535413321 1${lastid}="$place.3.54.1.3.32.1.1.$id"
      eval getnext_136141202125535413321 4${lastid}="$place.3.54.1.3.32.1.4.$id"
      eval getnext_136141202125535413321 13${lastid}="$place.3.54.1.3.32.1.13.$id"
      eval getnext_136141202125535413321 26${lastid}="$place.3.54.1.3.32.1.26.$id"
    fi

    rssi=$(wl rssi $mac | cut -d" " -f3)
    if test $rssi -eq 0
    then
      snr=0
    else
      let snr=-1*$noise_reference+$rssi
    fi
    mac=$(echo $mac | tr : ' ')

    eval value_136141202125535413321 1${id}=$id;
    eval type_136141202125535413321 1${id}='integer';
    eval value_136141202125535413321 4${id}='$mac';
    eval type_136141202125535413321 4${id}='octet';
    eval value_136141202125535413321 13${id}=$noise_reference;
    eval type_136141202125535413321 13${id}='integer';
    eval value_136141202125535413321 26${id}=$snr;
    eval type_136141202125535413321 26${id}='integer';
```

```
    lastid=$id
    let id=$id+1

  done

  if test $lastid -ne 0
  then
    eval getnext_13614120212553541332l1${lastid}="$place.3.54.1.3.32.1.4.1"
    eval getnext_13614120212553541332l4${lastid}="$place.3.54.1.3.32.1.13.1"
    eval getnext_136141202125535413321l3${lastid}="$place.3.54.1.3.32.1.26.1"
    eval getnext_136141202125535413321z6${lastid}="NONE"
  fi
}

LASTREFRESH=0

while read CMD
do
  case "$CMD" in
    PING)
      echo PONG
      continue
      ;;
    getnext)
      read REQ
      let REFRESH=$(date +%s)-$LASTREFRESH
      if test $REFRESH -gt 30
      then
        LASTREFRESH=$(date +%s)
        refresh
      fi

      oid=$(echo $REQ | tr -d .)
      eval ret=\$getnext_${oid}
      if test "x$ret" = "xNONE"
      then
        echo NONE
        continue
      fi
      ;;
    *)
      read REQ
      if test "x$REQ" = "x$place"
      then
        echo NONE
        continue
      else
```

```
      ret=$REQ
    fi
    ;;
  esac

  oid=$(echo $ret | tr -d .)
  if eval test "x\$type_${oid}" != "x"
  then
    echo $ret
    eval echo "\$type_${oid}"
    eval echo "\$value_${oid}"
  else
    echo NONE
  fi

done
```

2. Energy Usage and Wireless Users Code

2.1. Private OID

```
import time
from pysnmp.entity.rfc3413.oneliner import cmdgen
from ZabbixSender import ZabbixSender, ZabbixPacket
#import openpyxl
import csv
#from pyzabbix import ZabbixAPI
#getting information from the AP usingSNMP get regquest
def getsnmpinfo(value,ip,community):
    global info,res
    generator = cmdgen.CommandGenerator()
    comm_data = cmdgen.CommunityData('server', community, 1) # 1 means version
SNMP v2c
    #Class instance representing SNMP credentials.
    transport = cmdgen.UdpTransportTarget((ip, 161))
    real_fun = getattr(generator, 'nextCmd')
    res = (errorIndication, errorStatus, errorIndex, varBinds)\
        = real_fun(comm_data, transport, value)
    if not errorIndication is None  or errorStatus is True:
          print ("Error: %s %s %s %s" % res)
    else:
          x=0
          info=[]
          for varBindTableRow in varBinds:
              x=x+1
              for name, value in varBindTableRow:
                  info.append(value.prettyPrint()) #saving value into dat[]
    return
def sendinfoallowedMAC(Status,MAC,Item):
```

```
    packet = ZabbixPacket()
    packet.add(Item,Item, Status +' '+'MAC'+' '+ MAC,int(round(time.time())))
    server.send(packet)
    print(server.status,Status,MAC)
    return
with open('C:/Users/AndyNazario/Desktop/UPC Courses/MT/APIs/Offices.csv',
newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    reader=csv.reader(csvfile)
    data=list(reader)
OfficeN=[]
Ipadd=[]
Community=[]
for x in data:
    OfficeN.append(x[0])
for x in data:
    Ipadd.append(x[1])
for x in data:
    Community.append(x[2])
server = ZabbixSender('192.168.190.133', 10051) #Zabbix server
packet = ZabbixPacket()
for x in OfficeN[1:]:
    getsnmpinfo((1,3,6,1,4,1,2021,255,3,54,1,3,32,1,4),Ipadd[int(x)],Community[int(x)])
    if 'No SNMP response' in str(res):
        print('{}{}'.format('No SNMP response from',Ipadd[int(x)]))
    else:
        print('user(s) connected', info)
        Wiusers=len(info)#number of wireless users
        #ENERGY USAGE
        #sending 0 if no user is connected, sending 1 if a user is connected
        if Wiusers > 0 :
            packet.add('{}{}'.format('Office',x),'{}{}'.format('EnergyKeyOf',x),
'1',int(round(time.time())))
            #print('Energy ON','\n', server.status)
        else :
            packet.add('{}{}'.format('Office',x),'{}{}'.format('EnergyKeyOf',x),
'0',int(round(time.time())))
            #print('Energy OFF','\n', server.status)
            #Number of users connected to the wireless network
        packet.add('{}{}'.format('Office',x),'{}{}'.format('UsersKeyOf',x),
Wiusers,int(round(time.time())))
        server.send(packet)
        print(server.status)
```

2.2. Public OID

```
import time
from pysnmp.entity.rfc3413.oneliner import cmdgen
from ZabbixSender import ZabbixSender, ZabbixPacket
import csv
import ipaddress
def getsnmpinfo(value,ip,community,exMAC,IPnet,MASK):
    global res,info
    generator = cmdgen.CommandGenerator()
    comm_data = cmdgen.CommunityData('server', community, 1) # 1 means version
SNMP v2c
    #Class instance representing SNMP credentials.
    transport = cmdgen.UdpTransportTarget((ip, 161))
    real_fun = getattr(generator, 'nextCmd')
    res = (errorIndication, errorStatus, errorIndex, varBinds)\
        = real_fun(comm_data, transport, value)
    if not errorIndication is None  or errorStatus is True:
        print ("Error: %s %s %s %s" % res)
    else:
        x=0
        infot=[]
        for varBindTableRow in varBinds:
            x=x+1
            for name, value in varBindTableRow:
                infot.append(value.prettyPrint()) #saving value into dat[]
    #adding following lines to obtain only Wifi MAC users on info variable
    infoMAC=[k for k in infot if '0x' in k] # choosing only MACs from snmpwalk
    infoIP=[k for k in infot if '.' in k] # choosing only MACs from snmpwalk
    host = ipaddress.IPv4Address(IPnet)
    net = ipaddress.IPv4Network(IPnet + '/' + MASK, False)
    subnet=ipaddress.IPv4Address(int(host) & int(net.netmask))
    for I in infoIP:
        IP4=ipaddress.IPv4Address(I)
        if IP4 < subnet:
            infoMAC[infoIP.index(I)]=0
    if ' ' in exMAC:
        exMAC=exMAC.split()#separating the MACs
        for y in exMAC:#for each excluded MAC.
            if y in infoMAC:
                infoMAC.remove(y) #erasing MAC.
    else:
        if exMAC in infoMAC:
            infoMAC.remove(exMAC)
    infoMAC[:]=(value for value in infoMAC if value != 0)
    info=infoMAC
    return
def sendinfoallowedMAC(Status,MAC,Item):
    packet = ZabbixPacket()
    packet.add(Item,Item, Status +' '+'MAC'+' '+ MAC,int(round(time.time())))
```

```
    server.send(packet)
    print(server.status,Status,MAC)
    return
with open('C:/Users/AndyNazario/Desktop/UPC Courses/MT/APIs/Offices.csv',
newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    reader=csv.reader(csvfile)
    data=list(reader)

OfficeN=[]
Ipadd=[]
Community=[]
ExcludedMAC=[]
IPnet=[]
IPmask=[]
for x in data:
    OfficeN.append(x[0])
for x in data:
    Ipadd.append(x[1])
for x in data:
    Community.append(x[2])
for x in data:
    ExcludedMAC.append(x[3])
for x in data:
    IPnet.append(x[4])
for x in data:
    IPmask.append(x[5])
server = ZabbixSender('192.168.190.128', 10051) #Zabbix server
packet = ZabbixPacket()
for x in OfficeN[1:]:

getsnmpinfo((1,3,6,1,2,1,4,22),Ipadd[int(x)],Community[int(x)],ExcludedMAC[int(x)].lo
wer(),IPnet[int(x)],IPmask[int(x)])
    if 'No SNMP response' in str(res):
        print('{}{}'.format('No SNMP response from',Ipadd[int(x)]))
    else:
        print('user(s) connected', info)
        Wiusers=len(info)#number of wireless users
        #ENERGY USAGE
        #sending 0 if no user is connected, sending 1 if a user is connected
        if Wiusers > 0 :
            packet.add('{}{}'.format('Office',x),'{}{}'.format('EnergyKeyOf',x),
'1',int(round(time.time())))
            #print('Energy ON','\n', server.status)
        else :
            packet.add('{}{}'.format('Office',x),'{}{}'.format('EnergyKeyOf',x),
'0',int(round(time.time())))
            #print('Energy OFF','\n', server.status)
```

```
        #Number of users connected to the wireless network
      packet.add('{}{}'.format('Office',x),'{}{}'.format('UsersKeyOf',x),
Wiusers,int(round(time.time())))
      packet.add('{}{}'.format('Office',x),'{}{}'.format('MacsOf',x),
info,int(round(time.time())))
      server.send(packet)
      print(server.status)
```

3. Allowed Users Access Control Code

3.1. Private OID

```
#import sys
import time
from pysnmp.entity.rfc3413.oneliner import cmdgen
from ZabbixSender import ZabbixSender, ZabbixPacket
from pyzabbix import ZabbixAPI
import csv
def getsnmpinfo(value,ip,community):
   global info,res
   #ip='192.168.10.1'
   #community='public'
   generator = cmdgen.CommandGenerator()
   comm_data = cmdgen.CommunityData('server', community, 1) # 1 means version
SNMP v2c
   #Class instance representing SNMP credentials.
   transport = cmdgen.UdpTransportTarget((ip, 161))
   real_fun = getattr(generator, 'nextCmd')
   res = (errorIndication, errorStatus, errorIndex, varBinds)\
      = real_fun(comm_data, transport, value)
   if not errorIndication is None  or errorStatus is True:
        print ("Error: %s %s %s %s" % res)
   else:
        x=0
        info=[]
        for varBindTableRow in varBinds:
            x=x+1
            for name, value in varBindTableRow:
                info.append(value.prettyPrint()) #saving value into dat[]
   return
def sendinfoallowedMAC(MSG,Item):
   global packet
   packet = ZabbixPacket()
   packet.add('{}{}'.format('Office',Office),Item,MSG,int(round(time.time())))
   return
def itemid(nameitem):
   itemname=[]
```

```
    global index,item_id,hosts #defining global variables
    index=0
    items = zapi.item.get(output=["itemid", "name"])
    for x in items:
        itemname.append(x['name'])
    if nameitem in itemname:
        for x in itemname:
            if index <= len(itemname):
                if x==nameitem:
                  item_id=items[index]['itemid']
                  break
                else :
                   index=index+1
    else :
        item_id='does not exist'
        index='does not exist'
    return item_id
def lastvals(item_id,limt,type):
    global lastvalues
    lastvalues=zapi.history.get(
                itemids=item_id,
                sortfield="clock",
                sortorder='DESC',
                limit=limt,
                history=type
                )
    return

zapi=ZabbixAPI('http://192.168.190.133/zabbix',user='Admin',password='zabbix')
#zapi=ZabbixAPI('http://192.168.190.133/zabbix')
#zapi.login('Admin','zabbix')

with open('C:/Users/AndyNazario/Desktop/UPC Courses/MT/APIs/Offices.csv',
newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    reader=csv.reader(csvfile)
    data=list(reader)
OfficeN=[]
Ipadd=[]
Community=[]

for x in data:
    OfficeN.append(x[0])
for x in data:
    Ipadd.append(x[1])
for x in data:
    Community.append(x[2])
```

```
for Office in OfficeN[1:] :
   with open('C:/Users/AndyNazario/Desktop/UPC
Courses/MT/APIs/"{}{}{}'.format('Office_',Office,'.csv'), newline='') as csvfile:
      spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
      reader=csv.reader(csvfile)
      dataMAC= list(reader)
   itemid('{}{}'.format('UsersKeyOf',Office))
   lastvals(item_id,4,3)
   lastv=[]
   for x in lastvalues:
      lastv.append(x['value'])
   if lastv[0] == lastv[1] == lastv[2] == lastv[3] == lastv[4] == lastv[5] == lastv[6] ==
lastv[7] == lastv[8] == lastv[9] == lastv[10]:
      pass
   else:
      server = ZabbixSender('192.168.190.133', 10051) #Zabbix server
      packet = ZabbixPacket()

getsnmpinfo((1,3,6,1,4,1,2021,255,3,54,1,3,32,1,4),Ipadd[int(Office)],Community[int(
Office)])
      if info == []: #SNMP info from AP
        i=1
        for y in dataMAC[1:]:
           itemid('{}{}'.format('USER_',dataMAC[i][3]))
           lastvals(item_id,1,4)
           if 'Disconnected' in str(lastvalues):
              pass
           else:
              sendinfoallowedMAC(MSG='{}{}{}{}{}{}'.format('Allowed User
Disconnected ',dataMAC[i][0],' with Device ',dataMAC[i][2],' MAC =
',dataMAC[i][4]),Item=('{}{}'.format('USER_',dataMAC[i][3])))
              server.send(packet)
              print(server.status)
           i=i+1
      else:
         if (None,) in dataMAC:  #no ausers are registered in the file fro give Office
           pass
         else:
            i=1
            for y in dataMAC[1:]:
               if dataMAC[i][4].lower() in info:
                  itemid('{}{}'.format('USER_',dataMAC[i][3]))
                  lastvals(item_id,1,4)
                  if 'Connected' in str(lastvalues):
                     pass
                  else:
```

```
                    sendinfoallowedMAC(MSG='{}{}{}{}{}{}'.format('Allowed User
Connected ',dataMAC[i][0],' with Device ',dataMAC[i][2],' MAC =
',dataMAC[i][4]),Item=('{}{}'.format('USER_',dataMAC[i][3])))
                    server.send(packet)
                    print(server.status)
                if dataMAC[i][4].lower() not in info:
                    itemid('{}{}'.format('USER_',dataMAC[i][3]))
                    lastvals(item_id,1,4)
                    if 'Disconnected' in str(lastvalues):
                        pass
                    else:
                        sendinfoallowedMAC(MSG='{}{}{}{}{}{}'.format('Allowed User
Disconnected ',dataMAC[i][0],' with Device ',dataMAC[i][2],' MAC =
',dataMAC[i][4]),Item=('{}{}'.format('USER_',dataMAC[i][3])))
                        server.send(packet)
                        print(server.status)
                i=i+1
```

3.2. Public OID

```
import time
from pysnmp.entity.rfc3413.oneliner import cmdgen
from ZabbixSender import ZabbixSender, ZabbixPacket
from pyzabbix import ZabbixAPI
import csv
import ipaddress
def getsnmpinfo(value,ip,community,exMAC,IPnet,MASK):
    global info,res
    generator = cmdgen.CommandGenerator()
    comm_data = cmdgen.CommunityData('server', community, 1) # 1 means version
SNMP v2c
    #Class instance representing SNMP credentials.
    transport = cmdgen.UdpTransportTarget((ip, 161))
    real_fun = getattr(generator, 'nextCmd')
    res = (errorIndication, errorStatus, errorIndex, varBinds)\
        = real_fun(comm_data, transport, value)
    if not errorIndication is None  or errorStatus is True:
        print ("Error: %s %s %s %s" % res)
    else:
        x=0
        infot=[]
        for varBindTableRow in varBinds:
            x=x+1
            for name, value in varBindTableRow:
                infot.append(value.prettyPrint()) #saving value into dat[]
    #adding following lines to obtain only Wifi MAC users on info variable
    infoMAC=[k for k in infot if '0x' in k] # choosing only MACs from snmpwalk
```

```
    infoIP=[k for k in infot if '.' in k] # choosing only MACs from snmpwalk
    host = ipaddress.IPv4Address(IPnet)
    net = ipaddress.IPv4Network(IPnet + '/' + MASK, False)
    subnet=ipaddress.IPv4Address(int(host) & int(net.netmask))
    for I in infoIP:
        IP4=ipaddress.IPv4Address(I)
        if IP4 < subnet:
            infoMAC[infoIP.index(I)]=0
    if ' ' in exMAC:
        exMAC=exMAC.split()#separating the MACs
        for y in exMAC:#for each excluded MAC.
            if y in infoMAC:
                infoMAC.remove(y) #erasing MAC.
    else:
        if exMAC in infoMAC:
            infoMAC.remove(exMAC)
    infoMAC[:]=(value for value in infoMAC if value != 0)
    info=infoMAC
    return
def sendinfoallowedMAC(MSG,Item):
    global packet
    packet = ZabbixPacket()
    packet.add('{}{}'.format('Office',Office),Item,MSG,int(round(time.time())))
    return
def itemid(nameitem):
    itemname=[]
    global index,item_id,hosts #defining global variables
    index=0
    items = zapi.item.get(output=["itemid", "name"])
    for x in items:
        itemname.append(x['name'])
    if nameitem in itemname:
        for x in itemname:
            if index <= len(itemname):
                if x==nameitem:
                    item_id=items[index]['itemid']
                    break
                else :
                    index=index+1
    else :
        item_id='does not exist'
        index='does not exist'
    return item_id
def lastvals(item_id,limt,type):
    global lastvalues
    lastvalues=zapi.history.get(
                itemids=item_id,
                sortfield="clock",
```

```
            sortorder='DESC',
            limit=limt,
            history=type
            )
    return

zapi=ZabbixAPI('http://192.168.190.128/zabbix',user='Admin',password='zabbix')
#zapi=ZabbixAPI('http://192.168.190.133/zabbix')
#zapi.login('Admin','zabbix')

with open('C:/Users/AndyNazario/Desktop/UPC Courses/MT/APIs/Offices.csv',
newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    reader=csv.reader(csvfile)
    data=list(reader)
OfficeN=[]
Ipadd=[]
Community=[]
ExcludedMAC=[]
IPnet=[]
IPmask=[]
for x in data:
    OfficeN.append(x[0])
for x in data:
    Ipadd.append(x[1])
for x in data:
    Community.append(x[2])
for x in data:
    ExcludedMAC.append(x[3])
for x in data:
    IPnet.append(x[4])
for x in data:
    IPmask.append(x[5])
for Office in OfficeN[1:] :
    with open('C:/Users/AndyNazario/Desktop/UPC Courses/MT/APIs/Reg_Users.csv',
newline='') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
        reader=csv.reader(csvfile)
        dataMAC1= list(reader)
    itemid('{}{}'.format('MacsOf',Office))
    lastvals(item_id,3,1)
    lastv=[]
    for x in lastvalues:
        lastv.append(x['value'])
    if lastv[0] == lastv[1] == lastv[2] :
        pass
    else:
        server = ZabbixSender('192.168.190.128', 10051) #Zabbix server
```

```
        packet = ZabbixPacket()

getsnmpinfo((1,3,6,1,2,1,4,22),Ipadd[int(Office)],Community[int(Office)],ExcludedMA
C[int(Office)],IPnet[int(Office)],IPmask[int(Office)])
        dataMAC=[]
        for y in dataMAC1[1:]:
            if y[0][0] == Office:
                dataMAC.append(y)
        if info == []: #SNMP info from AP
            i=0
            for y in dataMAC[:]:
                itemid('{}{}'.format('USER_',dataMAC[i][4]))
                lastvals(item_id,1,4)
                if 'Disconnected' in str(lastvalues):
                    pass
                else:
                    sendinfoallowedMAC(MSG='{}{}{}{}{}{}'.format('Allowed User
Disconnected ',dataMAC[i][1],' with Device ',dataMAC[i][3],' MAC =
',dataMAC[i][5]),Item=('{}{}'.format('USER_',dataMAC[i][4])))
                    server.send(packet)
                    print(server.status)
                i=i+1
        else:
            if (None,) in dataMAC:  #no users are registered in the file fro given Office
                pass
            else:
                i=0
                for y in dataMAC[:]:
                    if dataMAC[i][5].lower() in info:
                        itemid('{}{}'.format('USER_',dataMAC[i][4]))
                        lastvals(item_id,1,4)
                        if 'Connected' in str(lastvalues):
                            pass
                        else:
                            sendinfoallowedMAC(MSG='{}{}{}{}{}{}'.format('Allowed User
Connected ',dataMAC[i][1],' with Device ',dataMAC[i][3],' MAC =
',dataMAC[i][5]),Item=('{}{}'.format('USER_',dataMAC[i][4])))
                            server.send(packet)
                            print(server.status)
                    if dataMAC[i][5].lower() not in info:
                        itemid('{}{}'.format('USER_',dataMAC[i][4]))
                        lastvals(item_id,1,4)
                        if 'Disconnected' in str(lastvalues):
                            pass
                        else:
                            sendinfoallowedMAC(MSG='{}{}{}{}{}{}'.format('Allowed User
Disconnected ',dataMAC[i][1],' with Device ',dataMAC[i][3],' MAC =
',dataMAC[i][5]),Item=('{}{}'.format('USER_',dataMAC[i][4])))
```

```
                server.send(packet)
                print(server.status)
            i=i+1
```

4. Python code for Automatic creation of Hosts, Items and Graphs

```
from pyzabbix import ZabbixAPI
import csv
def gethostid(hname):
    hostname=[]
    global index,host_id,hosts #defining global variables
    index=0
    hosts = zapi.host.get(output=["hostid", "name"])
    for x in hosts:
        hostname.append(x['name'])
    if hname in hostname:
        for x in hostname:
            if index <= len(hostname):
                if x==hname:
                   host_id=hosts[index]['hostid']
                   break
                else :
                    index=index+1
    else :
        host_id='does not exist'
        index='does not exist'
    return host_id
def itemid(nameitem):
    itemname=[]
    global index,item_id,hosts #defining global variables
    index=0
    items = zapi.item.get(output=["itemid", "name"])
    for x in items:
        itemname.append(x['name'])
    if nameitem in itemname:
        for x in itemname:
            if index <= len(itemname):
                if x==nameitem:
                   item_id=items[index]['itemid']
                   break
                else :
                    index=index+1
    else :
        item_id='does not exist'
        index='does not exist'
    return item_id
```

```
zapi=ZabbixAPI('http://192.168.190.128/zabbix',user='Admin',password='zabbix')
#zapi=ZabbixAPI('http://192.168.190.128/zabbix')
#zapi.login('Admin','zabbix')
with open('C:/Users/AndyNazario/Desktop/UPC Courses/MT/APIs/Offices.csv',
newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    reader=csv.reader(csvfile)
    data=list(reader)
    #print(data)
OfficeN=[]
Ipadd=[]
Community=[]
for x in data: #saving loaded data on a variable
    OfficeN.append(x[0])
for x in data: #saving loaded data on a variable
    Ipadd.append(x[1])

for x in OfficeN[1:]:
    gethostid('{}{}'.format('Office',x))
    if  index=='does not exist':#if not it creates a host and  an item
        zapi.host.create({
                "host": '{}{}'.format('Office',x),
                "interfaces": [
                    {"type": 1,'ip': Ipadd[int(x)],"port": 10050,"useip": 1,"main": 1,'dns':
"Energy"}
                    ],
                "groups": [{
                    "groupid": 5
                }],
                })
        gethostid('{}{}'.format('Office',x))
        zapi.item.create(
                hostid=host_id,
                name='{}{}'.format('EnergyKeyOf',x),
                key_='{}{}'.format('EnergyKeyOf',x),
                type=2,
                value_type=3,
                delay=30
                )

        zapi.item.create(
                    hostid=host_id,
                    name='{}{}'.format('UsersKeyOf',x),
                    key_='{}{}'.format('UsersKeyOf',x),
                    type=2,
                    value_type=3,
                    delay=30
                    )
```

```
        zapi.item.create(
                hostid=host_id,
                name='{}{}'.format('MacsOf',x),
                key_='{}{}'.format('MacsOf',x),
                type=2,
                value_type=1, #character
                delay=30
                )
        print('{}{}'.format('Office',x), 'host/items has been created')
        #Creates Zabbix Trapper item from the information given in the csv file.
    with open('C:/Users/AndyNazario/Desktop/UPC
Courses/MT/APIs/''{}{}{}'.format('Office_',x,'.csv'), newline='') as csvfile:
         spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
         reader=csv.reader(csvfile)
         dataMAC= list(reader)
    if dataMAC != [[]]:
        i=0
        for y in dataMAC[1:]:
            USER=dataMAC[1:][i][3]
            i=i+1
            if itemid('{}{}'.format('USER_',USER)) == 'does not exist':
                zapi.item.create(
                hostid=host_id,
                name='{}{}'.format('USER_',USER),
                key_='{}{}'.format('USER_',USER),
                type=2,
                value_type=4,
                delay=30
                )
                print('{}{}'.format('Office',x), 'User Trapper items has been created')
    itemid('{}{}'.format('EnergyKeyOf',x))
    zapi.graph.create({
        "name":'{}{}'.format('Energy Usage Office',x),
        "width":900,
        "height": 300,
        "graphtype":1,
        "gitems":
        [{"itemid": item_id,"type": 2,"color": "039900"}]
        })
    itemid('{}{}'.format('UsersKeyOf',x))
    zapi.graph.create({
        "name":'{}{}'.format('Wireless Users Ofice',x),
        "width":900,
        "height": 300,
        "graphtype":1,
        "gitems":
        [{"itemid": item_id,"type": 2,"color": "039900"}]
        })
```

```
print('{}{}'.format('Office',x), 'graph has been created')
```

5. Python code to extract data from Zabbix using API on Power BI software.

```
from pyzabbix import ZabbixAPI
import pandas as pd
def gethostid(hname):
    hostname=[]
    global index,host_id,hosts #defining global variables
    index=0
    hosts = zapi.host.get(output=["hostid", "name"])
    for x in hosts:
        hostname.append(x['name'])
    if hname in hostname:
        for x in hostname:
            if index <= len(hostname):
                if x==hname:
                  host_id=hosts[index]['hostid']
                  break
                else :
                   index=index+1
    else :
        host_id='does not exist'
        index='does not exist'
    return host_id
def lastvals(host_id,type):
    global values
    values=zapi.history.get(
                hostids=host_id,
                sortfield="clock",
                sortorder='DESC',
                #limit=limt,  # to limit the number of values analyzed
                history=type
                )
    return
zapi=ZabbixAPI('http://192.168.190.133/zabbix',user='Admin',password='zabbix')
#zapi=ZabbixAPI('http://192.168.190.133/zabbix')
#zapi.login('Admin','zabbix')

items = zapi.item.get(output=["itemid", "name"])
labels = ['itemid', 'name']
itemsids = pd.DataFrame.from_records(items, columns=labels)
gethostid('Office1')
lastvals(host_id,3)
labels = ['clock', 'itemid','ns','value']
Office1= pd.DataFrame.from_records(values, columns=labels)
```